Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«ОПТИМАЛЬНЕ КЕРУВАННЯ СИСТЕМАМИ МАСОВОГО ОБСЛУГОВУВАННЯ / OPTIMAL CONTROL OF QUEUEING SYSTEMS»**

Виконав: студент 4-го року навчання освітньої програми «Прикладна математика»,
спеціальності 113 Прикладна математика

Поліщук Максим Олександрович

Керівник: Чорней Р.К. доцент, кандидат фіз.-мат. наук

Рецензент:

_____

Кваліфікаційна робота захищена з оцінкою

_____

Секретар ЕК

_____

(підпис)

«_____» _____20___р.

Київ - 2025

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

ЗАТВЕРДЖУЮ
Зав.кафедри
математики, доцент, кандидат
фіз.-мат. наук
_____*Чорней Р.К.*
(підпис)
“____”_____2024

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
для кваліфікаційної роботи
студенту 4-го курсу, факультету
інформатики Поліщуку Максиму
Олександровичу

Тема: «Оптимальне керування системами масового обслуговування / Optimal Control of Queueing Systems»

Зміст кваліфікаційної роботи:
Анотація
Вступ
Розділ 1: Теоретична постановка задачі оптимального керування системами масового обслуговування
Розділ 2: Практична реалізація та приклад роботи
Висновки
Перелік використаних джерел

Дата видачі “____”_____2025 Керівник _____
(підпис)
Завдання отримав _____
(підпис)

ГРАФІК ПІДГОТОВКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Графік узгоджено «_____» _____2024р.

№ з/п	Перелік робіт	Термін виконання етапу	Підпис наукового керівника	Дата ознайомлення наукового керівника	Примітка
1.	Отримання теми квалі-фікаційної роботи.	10.10.2024			
2.	Ознайомлення з темою кваліфікаційної роботи.	05.11.2024			
3.	Розробка плану та структури роботи.	07.11.2024			
4.	Робота з науковою літературою, опис основних означень. Написання вступу та анотації.	07.11.2024			
5.	Дослідження основних підходів до розв’язання задач.	23.11.2024			
6.	Робота над текстовим оформленням теоретичної частини та одержаних результатів.	18.02.2025			
7.	Аналіз практичної частини, її корегування	29.03.2025			
8.	Попередній аналіз кваліфікаційної роботи. Виправлення помилок.	04.04.2025			
9.	Попередній захист кваліфікаційної роботи.	23.05.2025 5			
10.	Захист кваліфікаційної роботи.	04.06.2025			

Науковий керівник Чорней Руслан Костянтинович

Виконавець кваліфікаційної роботи Поліщук Максим Олександрович

ЗМІСТ

АНОТАЦІЯ

Кваліфікаційну роботу присвячено дослідженню оптимального керування системою масового обслуговування з дискретним часом. Така система моделюється ланцюгом Маркова з пуассонівським вхідним потоком заявок та експоненційним розподілом тривалості обслуговування. Метою роботи є розробка методів і алгоритму, що дозволяють максимізувати середній дохід функціонування цієї системи шляхом вибору оптимальної стратегії керування. Для досягнення поставленої мети використано апарат марковських процесів прийняття рішень та методи динамічного програмування (принцип оптимальності Беллмана). У роботі сформульовано математичну модель системи, визначено рівняння оптимальності для оцінювання функції цінності та запропоновано алгоритм пошуку оптимальної політики. Розроблений алгоритм реалізовано програмно і перевірено на модельному прикладі; в результаті отримано оптимальну стратегію керування та оцінку максимально можливого середнього доходу системи.

Ключові слова: система масового обслуговування, марковський процес, оптимальне керування, керування сервісами, керування заявками, рівняння Беллмана, оптимальна стратегія.

ВСТУП

У сучасному світі, де кількість запитів і задач для обробки постійно зростає, ефективне керування системами масового обслуговування набуває особливої актуальності. Правильне управління чергами та обслуговуванням дозволяє забезпечити безперервну і продуктивну роботу різноманітних сервісів (телекомунікаційних, обчислювальних, виробничих тощо), запобігаючи перевантаженню системи. Таким чином, розробка алгоритмів оптимізації керування чергами є актуальним завданням для підвищення ефективності роботи цих систем.

Одним із підходів до оптимального керування стохастичними системами обслуговування є використання марковських моделей. Марковські процеси дозволяють описати випадкову динаміку системи, а введення керованих дій приводить до моделі марковського процесу прийняття рішень (MDP), яка дає математичний апарат для пошуку оптимальної стратегії. Методи динамічного програмування (зокрема, рівняння Беллмана) застосовуються для обчислення оптимальної політики керування на основі критерію ефективності (наприклад, мінімізації середніх витрат).

Метою даної роботи є розробка математичної моделі та алгоритму оптимального керування системою масового обслуговування на основі марковського процесу, що максимізує середній дохід системи. Для досягнення цієї мети необхідно вирішити такі основні завдання:

1. Проаналізувати теоретичні основи: поняття випадкових величин і розподілів, марковських процесів, систем масового обслуговування, а також основи марковських процесів прийняття рішень (MDP).
2. Побудувати математичну модель досліджуваної системи масового обслуговування у вигляді MDP: визначити множину станів системи, можливі керуючі дії та параметри процесу (інтенсивність надходження заявок, розподіл часу обслуговування тощо).

3. Сформулювати критерій ефективності та на його основі визначити рівняння оптимальності (рівняння Беллмана) для оцінки функції цінності станів.
4. Розробити алгоритм пошуку оптимальної стратегії керування системою на основі отриманого рівняння Беллмана та реалізувати цей алгоритм програмно.
5. Перевірити роботу алгоритму на прикладі моделі системи: обчислити оптимальну стратегію керування для заданих параметрів та оцінити досягнутий максимальний середній дохід.

Об'єктом дослідження є системи масового обслуговування, що описуються марковськими процесами з дискретним часом та обмеженою чергою, де вхідний потік заявок має розподіл Пуассона, а час обслуговування розподілений експоненційно. Предметом дослідження є алгоритм знаходження оптимальної стратегії керування такою системою, який забезпечує максимізацію середнього доходу її функціонування. Для розв'язання поставлених задач у роботі використовуються методи теорії ймовірностей і випадкових процесів, теорії масового обслуговування та динамічного програмування.

Структурно робота складається зі вступу, двох розділів, висновків і списку використаних джерел. У першому розділі викладено теоретичні основи: наведено базові визначення (випадкова величина, експоненційний та пуассонівський розподіли, марковський процес, система масового обслуговування тощо) і детально описано постановку задачі оптимального керування (характеристики обраної системи, множини її станів і допустимих дій, показник ефективності та рівняння Беллмана). Другий розділ присвячено практичній реалізації запропонованого алгоритму: описано програмну імплементацію, продемонстровано роботу алгоритму на конкретному прикладі з заданими параметрами системи. За результатами розрахунків визначено оптимальну стратегію керування та знайдено значення максимального середнього доходу системи.

РОЗДІЛ 1: ТЕОРЕТИЧНА ПОСТАНОВКА ЗАДАЧІ ОПТИМАЛЬНОГО КЕРУВАННЯ СИСТЕМАМИ МАСОВОГО ОБСЛУГОВУВАННЯ

1.1 Основні визначення

Спочатку сформулюємо основні визначення та позначення, необхідні для подальшого розуміння матеріалу.

Означення 1.1.1. Множина випадкових величин $\xi = (\xi^t : t \in T)$, докладніше:

$$\xi = (\xi^t : (\Omega, F, Pr) \to (X, \mathcal{X}), t \in T),$$

де $T \neq \emptyset$, називається випадковим процесом. Тут (Ω, F, Pr) – імовірнісний простір, $(X, \mathcal{X})$ – простір станів процесу [2].

Марковський процес – випадковий процес, що характеризується марковською властивістю: майбутній стан процесу залежить тільки від його поточного стану і не залежить від послідовності попередніх станів. Якщо $\{X_t, t \in T\}$ – випадковий процес, то для нього виконується умова Маркова:

$$P\{\xi_{t+1} = j | \xi_t = i, \xi_{t-1} = i_{t-1}, \dots, \xi_0 = i_0\} = P\{\xi_{t+1} = j | \xi_t = i\}$$ [5].

Система масового обслуговування (СМО) – стохастична система, призначена для опрацювання потоку запитів (заявок) за допомогою одного чи кількох сервісів обслуговування [1]. Така система зазвичай включає: вхідний потік заявок (джерело вимог, що надходять на обслуговування), чергу (механізм накопичення заявок, які чекають обслуговування, якщо всі сервіси зайняті) та канали обслуговування (прилади, які виконують обслуговування

заявок). Основними характеристиками СМО є закони розподілу випадкового часу між надходженнями заявок та часу обслуговування заявок, а також кількість каналів обслуговування і місткість черги.

Марковський процес прийняття рішень (MDP) – це марковський стохастичний процес, в якому передбачена можливість керування, тобто вибору дії на кожному кроці з метою впливу на процес [1]. MDP є моделлю задачі прийняття послідовних рішень в умовах невизначеності, що визначається наступними компонентами: станами, діями, імовірностями переходів та функцією винагороди (або витрат). На кожному кроці часу система знаходиться в деякому стані і приймає керуючу дію; у результаті дія впливає на перехід системи в новий стан і приносить певну винагороду або навпаки витрати.

1.2 Постановка задачі

У цій роботі я розглядатиму стохастичну систему масового обслуговування з обмеженою чергою та дискретними типами заявок. Потік заявок розподілений за законом Пуассона, час обслуговування кожної заявки є експоненціальним. Модель має такі параметри:

1. λ – інтенсивність надходження заявок;
2. c – кількість сервісів у системі;
3. K – максимальна довжина черги;
4. $N = K + c$ – повна місткість системи;
5. μ_i – інтенсивність обслуговування на i-му сервісі;
6. C_i – вартість утримання i-го сервіса активним за одиницю часу;
7. S_i – одноразова витрата за активацію i-го сервіса;
8. C_{wait} – вартість очікування однієї заявки в черзі за одиницю часу;

9. Типи заявок: припустимо, що заявки, які надходять, поділяються на скінченну множину типів $j = 1, 2, \ldots$ Заявка типу j приносить дохід R_j після її обслуговування. Ймовірність, що наступна заявка буде типу j, позначимо p_j, при цьому $\sum_1^j p_j = 1$.

Якщо на момент надходження нової заявки є вільний активний сервіс, заявка одразу береться на обслуговування. Якщо всі активні сервіси зайняті, заявка може бути додана у чергу (якщо довжина черги менша K) або бути відхилена. Після завершення обслуговування заявки на будь-якому сервісі цей сервіс звільняється і може прийняти наступну заявку з черги, якщо черга не порожня.

Динамічно керувати системою можна двома способами:

1. Активація/деактивація сервісів. Передбачаємо, що не всі c сервісів мусять бути постійно ввімкненими. Адміністратор системи може вмикати додаткові сервіси або вимикати діючі залежно від стану системи. Активація нового сервісу дозволяє збільшити пропускну здатність системи, але при цьому супроводжується одноразовою витратою S_i та постійними витратами C_i за його утримання. Вимкнення сервісу, навпаки, економить витрати на його утримання.
2. Прийняття або відхилення заявок. Оскільки система має обмежену місткість з втратами за очікування заявки в черзі C_{wait} і різні типи заявок приносять різний дохід, допускається політика відхилення деяких заявок. Коли надходить нова заявка, диспетчер системи може прийняти її, якщо є можливість відправити на сервіс чи в чергу, або відхилити, якщо вона не принесе дохід через більші витрати за очікування в черзі та за обслуговування. Відхилена заявка залишає систему без отримання доходу, але й без додаткових витрат за її очікування в черзі. Можливість відмовляти в обслуговуванні заявки дозволяє потенційно підвищити середній дохід системи.

Отже, система є марковською керованою системою масового обслуговування. Її динаміка описується марковським процесом з рішеннями, де стан змінюється при надходженні заявок та завершенням їх обслуговуванння, а в моменти таких змін приймаються керуючі дії. Метою керування є вибір оптимальної стратегії, щоб максимізувати довгостроковий середній прибуток системи.

1.3 Простір станів системи

Стан системи в довільний момент часу визначається станом черги (скільки заявок очікують на обслуговування) та станом кожного з сервісів (чи акимвний він і чи зайнятий обслуговуванням). Введемо $Q(t)$ – кількість заявок у черзі, що очікують на обслуговування, $0 \leq Q \leq K$, і $D(t)$ – вектор станів сервісів. $D(t) = (d_1, d_2, \dots, d_c)$, де компонента d_i відповідає i-му сервісу. Кожен d_i є впорядкованою парою показників $d_i = \left(d_i^{(1)}, d_i^{(2)}\right)$, де:

1. $d_i^{(1)} \in \{0, 1\}$ – індикатор того, чи сервіс i активний (1 – активний, 0 – неактивний);
2. $d_i^{(2)} \in \{0, 1\}$ – індикатор того, чи сервіс i зайнятий обслуговуванням заявки в даний момент (1 – сервіс обслуговує заявку, 0 – сервіс вільний).

Оскільки вимкнений сервіс не може обслуговувати заявки, для кожного виконується умова: якщо $d_i^{(1)} = 0$, то обов'язково $d_i^{(2)} = 0$.

Множину всіх станів системи позначимо через S. Усі допустимі стани можна описати так:

$$S = \left\{ (Q, d_1, \dots, d_c) \;\middle|\; \begin{array}{c} Q \in \{0, 1, \dots, K\}, \\ d_i = \left(d_i^{(1)}, d_i^{(2)}\right), i = 1, \dots, c, \\ d_i^{(1)} \in \{0, 1\}, d_i^{(2)} \in \{0, 1\} \\ d_i^{(2)} \leq d_i^{(1)}, i = 1, \dots, c \end{array} \right\}.$$

Тепер потрібно описати множину можливих дій, які можна застосувати в цьому стані, $A(s)$. Як було зазначено раніше, керування здійснюється двома способами: зміна набору активних сервісів та прийняття чи відхилення нових заявок. Опишемо компоненти дії:

1. Керування сервісами:

   Дію щодо сервісів можна задати як вибір підмножини сервісів, які мають бути активними після застосування дії. Відповідно, управління сервісами – це бінарний вектор $u = (u_1, u_2, \dots, u_c)$, де $u_i \in \{0,1\}$, який визначає новий стан сервісів (1 – сервіс i залишити активним або активувати, 0 – вимкнути або залишити вимкненим сервіс i).

   Але не всі вектори u є допустимими. Якщо в поточному стані сервіс зайнятий ($d_i^{(2)} = 1$), то він мусить залишитися активним ($u_i = 1$), оскільки ми не можемо вимкнути сервіс, поки він обслуговує заявку. Активація нового сервіса (перехід від $d_i^{(1)} = 0$ до $u_i = 1$) спричиняє витрати S_i, але ввімкнення дозволяється завжди.

   Таким чином, множина можливих керувань сервісів у стані s включає всі такі набори u, що $u_i \geq d_i^{(2)}$ для всіх i. Зокрема, дія «не змінювати склад активних сервісів» також належить цій множині.

2. Керування заявками:

   Коли надходить нова заявка і система перебуває в стані s, диспетчер має прийняти рішення щодо цієї заявки. Якщо система переповнена або якщо очікується, що обслуговування цієї заявки невигідне (наприклад, низький прибуток R_j і великі потенційні витрати), то заявка відхиляється і не потрапляє у систему. Інакше заявка приймається.

Дії щодо сервісів і заявки можуть здійснюватися одночасно. Наприклад, при надходженні заявки керуюча дія $a \in A$ може складатися з двох компонент: ми вирішуємо ввімкнути певний набір сервісів та вирішуємо, що робити з

поточною заявкою. Для стислості будемо позначати складну дію одним символом a, розуміючи, що вона може нести інформацію про обидва аспекти. Таким чином, множина допустимих дій $A(s)$ у стані s включає всі комбіновані рішення виду: вибір нового набору активних снрвісів $u = (u_1, u_2, \dots, u_c)$, де $u_i \in \{0,1\}$ та $u_i \geq d_i^{(2)}$ для всіх i, а також вибір опції щодо нової заявки, відхилити або прийняти, якщо $Q < K$.

1.4 Функція миттєвої винагороди

Необхідно визначити функцію миттєвої винагороди для кожного стану s і дії a. Ця функція відображає корисність (прибуток мінус витрати), яку отримає система при застосуванні дії a в стані s.

Якщо внаслідок переходу із стану s система завершує обслуговування певних заявок, то вона отримує відповідний прибуток. Зокрема, коли обслуговування заявки типу j на якомусь сервісі завершується, система заробляє суму R_j.

Кожен увімкнений сервіс (незалежно від того, зайнятий він чи простоює) генерує витрати C_i за одиницю часу. Тоді поточні витрати на їх роботу дорівнюють:

$$\sum_{i:d_i^{(1)}=1} C_i$$

в розрахунку на одиницю часу.

За кожну заявку в черзі система несе «штраф» C_{wait} за одиницю часу:

$$Q \cdot C_{wait}.$$

Цей компонент показує небажаність великих черг і затримок для заявок.

Якщо дія a передбачає ввімкнення деяких раніше вимкнених сервісів, то за кожен такий сервіс додається разова витрата S_i. Цю витрату слід враховувати миттєвою, в момент застосування дії.

Об’єднавши ці складові, можемо записати винагороду за стан s і дію a як:

$$R(s,a) = \sum_j p_j R_j - \sum_{i:d_i^{(1)}=1} C_i - Q \cdot C_{wait} - \sum_{i \in I_c} S_i.$$

Тут I_c – множина тих сервісів, які були вимкнені в стані s і увімкнені дією a. Зауважимо, що перші три складові залежать від стану і характеризують поточний дохід потоку часу, тоді як остання складова одноразова витрата, зумовлена дією. Для коректності врахування в рівняннях динамічного програмування, перші три складові зазвичай трактуються як ставка винагороди в стані, а при обчисленнях множаться на тривалість перебування в стані. В нашій моделі тривалість перебування в стані до наступної події є випадковою величиною, що залежить від сумарної інтенсивності подій. Надалі, при виведенні рівняння Беллмана, ми врахуємо це через ймовірності переходів і усереднення.

Наша мета – максимізувати середній прибуток $R(s,a)$.

1.5 Рівняння оптимальності

Задача визначення оптимальної стратегії керування формалізується як задача оптимального керованого марковського процесу. Ми шукаємо таку політику (правило вибору дій), яка максимізує середній дохід системи в довгостроковому періоді. Для розв'язання такої задачі застосовуються методи динамічного програмування. Центральним співвідношенням є рівняння Беллмана оптимальності.

Позначимо через π деяку стаціонарну стратегію, а через G_π середній дохід за одиницю часу при дотриманні стратегії π. Нехай $V_\pi(s)$ – довгострокова

середня цінність стану s при стратегії π – тобто очікувана функція виграшу, що показує, наскільки вигідно опинятися в стані s під цією стратегією. В оптимальній стратегії π^* ці значення задовольняють рівняння оптимальності. У нашій моделі з середнім критерієм оптимізації (без дисконтування) це рівняння можна записати так:

$$g^* + h(s) = \max_{a \in A(s)} \{R(s,a) + \sum_{s' \in S} P(s'|s,a)h(s')\}, \forall s \in S.$$

Це рівняння Беллмана для середньої винагороди [4].

Тут:

1. g^* – оптимальний середній дохід системи на одиницю часу (константа, яку ми хочемо максимізувати);
2. $h(s)$ – так звана диференційна цінність стану s, яка характеризує відносну вигоду перебування в стані s порівняно з середнім. Функція $h(s)$ разом з константою g^* є розв'язком рівняння Беллмана. Її можна інтерпретувати як надлишковий дохід, який отримаємо, починаючи з s, порівняно із середнім g^*, за нескінченно довгий час роботи;
3. $P(s'|s,a)$ – ймовірність переходу зі стану s в стан s' за один крок процесу, якщо в s застосована дія a. Сума $\sum_{s' \in S} P(s'|s,a)h(s')$ – це очікувана майбутня цінність стану після здійснення переходу. Цей доданок враховує всі можливі випадкові зміни стану (надходження заявок різних типів, завершення обслуговувань тощо) після дії a.

Формулою вище записана умова, що в оптимальній стратегії вибір дії aв стані s повинен максимізувати суму поточної миттєвої винагороди плюс очікуваної майбутньої цінності. Зліва стоїть $g^* + h(s)$ – це по суті очікувана винагорода за одиницю часу (g^*) плюс поправка цінності стану. Для оптимальної політики π^* рівняння має виконуватися як рівність при тій дії, що вибирає π^*. Якщо ж для якогось s дія a не оптимальна, то значення в фігурних

дужках для a буде нижчим за максимум.

Рівняння Беллмана є математичною формалізацією принципу оптимальності Беллмана, яке стверджує, що оптимальна стратегія володіє властивістю незмінності щодо хвоста: незалежно від початкового стану, подальші рішення мають становити оптимальну стратегію для процесу, що починається з нового стану. Для нашої системи рівняння Беллмана можна розписати більш детально, врахувавши структуру $R(s,a)$ та ймовірності переходів. Оскільки процес відбувається в безперервному часі, зручно застосувати метод уніформізації. Нехай $\delta = \lambda + \sum_{i:d_i^{(2)}=1} \mu_i$ – сумарна інтенсивність виходу з стану s (тобто сума інтенсивності надходження заявки та інтенсивностей завершення обслуговувань на кожному зайнятому сервісі). Тоді протягом нескінченно малого інтервалу dt відбудеться подія (або надходження, або завершення) з імовірністю δdt, а з імовірністю $1 - \delta dt$ стан не зміниться.

В такому підході рівняння оптимальності можна переписати еквівалентно як:

$$h(s) = \max_{a \in A(s)} \left\{ R(s,a)\frac{1}{\delta} + \frac{\lambda}{\delta} h\big(s_{arr}(a)\big) + \sum_{i:d_i^{(2)}=1} \frac{\mu_i}{\delta} h\left(s_{dep,i}(a)\right) - \frac{g^*}{\delta} \right\},$$

де $s_{arr}(a)$ – стан після надходження заявки (з урахуванням дії a, зокрема прийняти чи відмовити), $s_{dep,i}(a)$ – стан після завершення обслуговування на i-му сервісі (з урахуванням дії щодо можливого вимкнення цього сервісу тощо), а δ залежить від s (для кожного стану свої δ). Ця форма рівняння еквівалентна попередній і може бути отримана шляхом прирівнювання похідної функції вартості до нуля. Далі користуватимемося компактнішою формою рівняння Беллмана, розуміючи його зміст, наведений вище.

1.6 Оптимальна стратегія

Стратегія (політика) керування визначає, яку дію слід обирати в кожному можливому стані системи. Формально стратегія π може задаватися як відображення $\pi : S \rightarrow A$, що кожному стану s ставить у відповідність деяку дію $a \in A(s)$. Якщо стратегія детермінована (не випадкова) і не залежить явно від часу, то це стаціонарна детермінована політика. В рамках поставленої задачі достатньо розглядати саме стаціонарні детерміновані стратегії.

Оптимальна стратегія π^* – це стратегія, яка в кожному стані s вибирає дію, що максимізує праву частину рівняння Беллмана. Іншими словами, $\pi^*(s)$ – це така дія (або одна з таких дій, якщо максимум досягається не в єдиній точці), для якої досягається максимум у формулі з попереднього пункту. Формально можна записати:

$$\pi^*(s) \in arg \max_{a \in A(s)} \{R(s,a) + \sum_{s' \in S} P(s'|s,a)h(s')\}, \forall s \in S.$$

Оптимальна стратегія може бути не єдиною, але всі оптимальні стратегії дають однаковий середній дохід g^*. Знайшовши розв'язок рівняння Беллмана (попереднього пункту), ми тим самим знайдемо і π^*: для кожного s достатньо вибрати дію, яка досягає максимуму в тому рівнянні. Таким чином, π^* визначається як жадібна стратегія відносно оптимальної функції цінності $h(s)$ [4].

В нашій задачі оптимальна стратегія інтуїтивно повинна врівноважувати два фактори:

1. Політика управління чергою: оптимальна π^*, ймовірно, відхиляє ті заявки, які приносять недостатній чистий дохід або які можуть спричинити занадто великі витрати через чергу. Часто в подібних системах оптимальна політика має пороговий характер: існує деякий поріг для довжини черги чи зайнятості системи, вище якого нові заявки відхиляються.

2. Політика управління сервісами: оптимальна π^* повинна вирішувати, скільки сервісів тримати активними залежно від поточного навантаження. Очікується теж порогова структура: наприклад стратегія типу N-policy відома для подібних задач – сервіси вимикаються коли система порожня, і вмикаються лише коли накопичиться певне число заявок.

РОЗДІЛ 2: ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ПРИКЛАД РОБОТИ

2.1 Алгоритм Value Iteration для пошуку оптимальної стратегії

Одним з базових методів обчислення оптимальної стратегії в марковських задачах керування є метод ітерації значень (Value Iteration). Цей алгоритм ґрунтується на послідовному наближенні до оптимального значення функції $h(s)$ (або $V(s)$ в позначеннях теорії) за допомогою рекурентного застосування оператора Беллмана. Інтуїтивно, починаючи з деякого припущення про значення станів, алгоритм поступово уточнює ці значення, поки вони не збіжаться до оптимальних, після чого з отриманої функції цінностей легко побудувати оптимальну стратегію.

Опишемо кроки алгоритму:

Крок 0 (ініціалізація): Встановити початкову наближену функцію вартості $V_0(s)$ для всіх $s \in S$. Наприклад, можна взяти початкове наближення нульовим для всіх станів: $V_0(s) = 0, \forall s$. Нехай $\varepsilon > 0$ – задана точність (критерій зупинки).

Крок 1 (ітераційне оновлення): Для $n = 0,1,2,\ldots$ обчислюємо нову функцію $V_{n+1}(s)$ для всіх станів s за оператором Беллмана:

$$V_{n+1}(s) = \max_{a \in A(s)} \{R(s,a) + \sum_{s' \in S} P(s'|s,a) V_n(s')\}.$$

Це рекурентне співвідношення оновлює оцінку цінності стану s шляхом вибору найкращої дії і використання поточних оцінок $V_n(s')$ для наступних станів. По суті, це дискретна форма рівняння Беллмана оптимальності (без члену g, якщо вважати, що наявність V_n вже частково врахувала його або працюємо в задачі з дисконтом).

На практиці, оскільки наш критерій – середній дохід, можна додатково застосовувати нормалізацію після кожного кроку оновлення, щоб значення V_n не дрейфували на константу. Наприклад, зручно фіксувати $V_{n+1}(s_0) = 0$ для

деякого базового стану s_0 на кожній ітерації.

Крок 2 (перевірка збіжності): Після кожної ітерації перевіряємо наскільки змінились значення функції: обчислюємо $\Delta_n = \max_{s \in S} |V_{n+1}(s) - V_n(s)|$.

Це максимальна різниця між оцінками на сусідніх ітераціях. Якщо $\Delta_n < \varepsilon$, тобто зміни дуже малі, алгоритм вважається збіжним і переходимо до кроку 3. Інакше повторюємо крок 1 для чергової ітерації.

Крок 3 (побудова політики): Коли отримано наближення $V_N(s)$, близьке до оптимального $h(s)$, можна визначити наближену оптимальну стратегію. Для кожного стану s обираємо дію, що реалізує максимум у правій частині (на основі останньої ітерації значень):

$$\pi^*(s) \in arg \max_{a \in A(s)} \{R(s,a) + \sum_{s' \in S} P(s'|s,a) V_N(s')\}.$$

Іншими словами, ми робимо політику жадібною до знайденої оцінки функції вартості. Отримана таким чином стратегія буде оптимальною (якщо алгоритм досягнув точної збіжності) або ε-оптимальною (якщо зупинено при малому ε).

Зупинка алгоритму: Формальний критерій ми задали як $\Delta_n < \varepsilon$. Також можуть використовуватися альтернативні критерії, наприклад, перевірка, що для кожного стану s виконується майже рівність при деякій дії (тобто різниця між лівою і правою частиною рівняння Беллмана менша за ε).

Метод ітерації значень є доволі універсальним і гарантує знаходження оптимальної стратегії π^*.

2.2 Опис створеної програми

Розроблено програму мовою Python, що моделює процес обслуговування заявок із керуванням сервісами. Ця програма реалізує алгоритм пошуку оптимальної стратегії керування системою масового обслуговування на основі

марковської моделі і рівняння Беллмана.

При запуску програми спершу викликається функція read_parameters(). Вона читає вхідні параметри моделі (інтенсивність потоку заявок λ, інтенсивність обслуговування кожного сервісу μ_i, максимальну довжину черги K, витрати на утримання сервісів C_i, витрати за запуск сервісів S_i, витрати за очікування заявки в черзі C_{wait} і параметри різних типів заявок: R_j і p_j) та формує на їх основі початкові дані для моделювання. Функція generate_states(c, K) генерує множину всіх можливих станів системи, тобто всіх комбінацій, які може набувати система за кількістю заявок у черзі та кількістю активних сервісів.

Для кожного стану визначено набір можливих керуючих дій, що моделюються функцією generate_actions(s, K). Ця функція повертає множину допустимих дій у стані s. Дії охоплюють рішення щодо кількості сервісів, які мають бути активними, а також політику прийому заявок. Наприклад, дія може передбачати: залишити поточну кількість сервісів увімкненою, або ввімкнути додатковий сервіс, або вимкнути зайвий сервіс, і одночасно визначити, чи приймати вхідну заявку у випадку її надходження. Таким чином, generate_actions() відображає всі раціональні варіанти керування системою в кожному можливому стані.

Функція reward(s, a, C_i, S_i, C_wait, R_j, p_j) обчислює безпосередню винагороду при виборі певної дії в заданому стані. Це відповідає функції миттєвого виграшу моделі MDP. Зокрема, тут враховуються такі компоненти: дохід від обслуговування заявок і витрати на утримання сервісів. Якщо в результаті дії система приймає нову заявку, то додається дохід за її обслуговування. Якщо дією передбачається відмова в обслуговуванні (відхилення заявки), то система не отримує цього доходу. Також reward() враховує витрати на активні сервіси: за кожен увімкнений сервіс за одиницю часу стягуються експлуатаційні витрати. Окрім того, у разі наявності додаткових компонент витрат, ця функція включає їх: так, за наявності одноразової вартості активації сервісу, її додають як витрату при дії увімкнення

нового сервіса: якщо враховується вартість очікування в черзі, то при великій черзі виникають витрати пропорційні числу заявок, що очікують.

Функція probabilities_sa(s, a, l, mu) для поточного стану s і вибраної дії a визначає розподіл ймовірностей переходу в можливі наступні стани s'. Ці ймовірності враховують випадковий характер надходження заявок та завершення обслуговування. Вона формує всі можливі переходи: наприклад, з імовірністю p відбудеться прибуття нової заявки (якщо система не переповнена, вона перейде в стан з чергою на 1 заявку більшою; якщо було вирішено не приймати більше заявок або досягнуто межу K, нова заявка відхиляється і стан не зміниться), з імовірністю q відбудеться завершення обслуговування (система перейде до стану з чергою на 1 заявку меншою, причому функція filling_services(Q, D) одразу ж забезпечить призначення наступної заявки з черги на звільнений сервіс, якщо в черзі хтось очікує), а з рештою ймовірності (1-p-q) стан залишиться тим самим.

Найважливішим компонентом програми є функція value_iteration(s, a_map, l, mu, C_i, S_i, C_wait, R_j, p_j, eps=1e-6, max_iter=10000), що реалізує ітераційний алгоритм пошуку оптимальної стратегії. Цей алгоритм базується на методі відносної функції цінності для марковських процесів із середнім доходом. Функція реалізовує алгоритм, що був описаний у підрозділі 2.1.

Програма перебирає всі стани і для кожного через probabilities_sa() визначає очікувані значення $V(s')$ наступних станів, додає відповідний reward та обирає дію, що дає максимум. Після багатьох ітерацій обчислення, значення $V(s)$ наближаються до стаціонарних. Коли досягнуто потрібної точності збіжності ε, алгоритм зупиняється. Функція value_iteration() повертає два результати: оптимальну стратегію для кожного стану s та оцінку середнього максимального доходу g.

2.3 Приклад роботи

Розглянемо конкретний приклад параметрів системи та побачимо результати, які видає програма. Нехай вхідний потік заявок має інтенсивність λ = 1.2, маємо 2 сервіси, середня швидкість обслуговування на сервісах $\mu = (0.8, 1.2)$, постійні витрати на сервіси $\mathrm{C} = (1.5,\ 2)$, разові витрати при запуску сервісів $S = (4,\ 6)$, максимальна довжина черги $K\ =\ 2$, втрати за очікування заявок в черзі $C_{wait} = 1$ і є два типи заявок з доходами $R = (6,\ 8)$ та ймовірностями $p = (0.6,\ 0.4)$.

Програма створила такі можливі стани:

[(0, ((0, 0), (0, 0))), (0, ((0, 0), (1, 0))), (0, ((0, 0), (1, 1))), (0, ((1, 0), (0, 0))), (0, ((1, 0), (1, 0))), (0, ((1, 0), (1, 1))), (0, ((1, 1), (0, 0))), (0, ((1, 1), (1, 0))), (0, ((1, 1), (1, 1))), (1, ((0, 0), (1, 0))), (1, ((0, 0), (1, 1))), (1, ((1, 0), (0, 0))), (1, ((1, 0), (1, 0))), (1, ((1, 0), (1, 1))), (1, ((1, 1), (0, 0))), (1, ((1, 1), (1, 0))), (1, ((1, 1), (1, 1))), (2, ((0, 0), (1, 0))), (2, ((0, 0), (1, 1))), (2, ((1, 0), (0, 0))), (2, ((1, 0), (1, 0))), (2, ((1, 0), (1, 1))), (2, ((1, 1), (0, 0))), (2, ((1, 1), (1, 0))), (2, ((1, 1), (1, 1)))];

і такі можливі дії:

{(0, ((0, 0), (0, 0))): [((0, 1), 'accept'), ((1, 0), 'accept'), ((1, 1), 'accept')], (0, ((0, 0), (1, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (0, ((0, 0), (1, 1))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (0, ((1, 0), (0, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (0, ((1, 0), (1, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (0, ((1, 0), (1, 1))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (0, ((1, 1), (0, 0))): [((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (0, ((1, 1), (1, 0))): [((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (0, ((1, 1), (1, 1))): [((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((0, 0), (1, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((0, 0), (1, 1))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((1, 0), (0, 0))): [((0, 1),

'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((1, 0), (1, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((1, 0), (1, 1))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((1, 1), (0, 0))): [((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((1, 1), (1, 0))): [((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (1, ((1, 1), (1, 1))): [((1, 1), 'accept'), ((1, 1), 'reject')], (2, ((0, 0), (1, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (2, ((0, 0), (1, 1))): [((0, 1), 'reject'), ((1, 1), 'reject')], (2, ((1, 0), (0, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (2, ((1, 0), (1, 0))): [((0, 1), 'accept'), ((0, 1), 'reject'), ((1, 0), 'accept'), ((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (2, ((1, 0), (1, 1))): [((0, 1), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (2, ((1, 1), (0, 0))): [((1, 0), 'reject'), ((1, 1), 'reject')], (2, ((1, 1), (1, 0))): [((1, 0), 'reject'), ((1, 1), 'accept'), ((1, 1), 'reject')], (2, ((1, 1), (1, 1))): [((1, 1), 'reject')]}.

Для кожного стану програма ініціалізувала $V(s)$ нулями, викликала функцію value_iteration() і почала свою роботу. Після завершення роботи ми отримали оптимальну стратегію з середнім максимальним доходом 4.7:

Стан (Q=0, D=((0, 0), (0, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=0, D=((0, 0), (1, 0))) → u=(0, 1), рішення: прийняти заявку

Стан (Q=0, D=((0, 0), (1, 1))) → u=(0, 1), рішення: прийняти заявку

Стан (Q=0, D=((1, 0), (0, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=0, D=((1, 0), (1, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=0, D=((1, 0), (1, 1))) → u=(1, 1), рішення: прийняти заявку

Стан (Q=0, D=((1, 1), (0, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=0, D=((1, 1), (1, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=0, D=((1, 1), (1, 1))) → u=(1, 1), рішення: прийняти заявку

Стан (Q=1, D=((0, 0), (1, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=1, D=((0, 0), (1, 1))) → u=(0, 1), рішення: прийняти заявку

Стан (Q=1, D=((1, 0), (0, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=1, D=((1, 0), (1, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=1, D=((1, 0), (1, 1))) → u=(1, 1), рішення: прийняти заявку

Стан (Q=1, D=((1, 1), (0, 0))) → u=(1, 0), рішення: прийняти заявку

Стан (Q=1, D=((1, 1), (1, 0))) → u=(1, 1), рішення: прийняти заявку

Стан (Q=1, D=((1, 1), (1, 1))) → u=(1, 1), рішення: прийняти заявку

Стан (Q=2, D=((0, 0), (1, 0))) → u=(1, 0), рішення: відхилити заявку

Стан (Q=2, D=((0, 0), (1, 1))) → u=(0, 1), рішення: відхилити заявку

Стан (Q=2, D=((1, 0), (0, 0))) → u=(1, 0), рішення: відхилити заявку

Стан (Q=2, D=((1, 0), (1, 0))) → u=(1, 0), рішення: відхилити заявку

Стан (Q=2, D=((1, 0), (1, 1))) → u=(1, 1), рішення: відхилити заявку

Стан (Q=2, D=((1, 1), (0, 0))) → u=(1, 0), рішення: відхилити заявку

Стан (Q=2, D=((1, 1), (1, 0))) → u=(1, 1), рішення: відхилити заявку

Стан (Q=2, D=((1, 1), (1, 1))) → u=(1, 1), рішення: відхилити заявку

ВИСНОВКИ

У даній кваліфікаційній роботі було розглянуто задачу оптимального керування системою масового обслуговування із заявками, чергою та можливістю динамічного керування сервісами. Основна мета полягала у побудові математичної моделі, що дозволяє максимізувати середній прибуток системи, та реалізації алгоритму для знаходження оптимальної стратегії прийняття рішень.

У теоретичній частині було сформульовано математичну постановку задачі: описано простір станів, множину дій, правила переходу між станами та функцію миттєвої винагороди. Сформульовано критерій оптимальності у вигляді середнього доходу за одиницю часу. Описано алгоритм відносної ітерації значень, який забезпечує знаходження політики оптимального керування у дискретному просторі.

У практичній частині було реалізовано програму мовою Python, що виконує обчислення оптимальної стратегії за допомогою згаданого алгоритму. Програма генерує всі допустимі стани системи, перелік можливих дій, оцінює ймовірності переходів і винагороди, а також дозволяє вивести оптимальну політику керування для заданих параметрів. На конкретному прикладі підтверджено працездатність рішення: алгоритм визначив оптимальні дії для різних станів системи (ввімкнення/вимкнення сервісів, прийняття/відхилення заявок) і оцінив максимальний середній прибуток.

Наступні дослідження можуть включати оптимізацію алгоритму, розширення моделі на випадок змінних інтенсивностей заявок, застосування машинного навчання для апроксимації стратегії в системах із великим простором станів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Badian-Pessot P., Down D.G., Lewis M.E. Optimal control policies for an M/M/1 queue with a removable server and dynamic service rates [Електронний ресуррс] – Режим доступу до ресурсу: https://people.orie.cornell.edu/plb93/Single-server-rate-control.pdf.
2. Chornei R. K., Daduna H., Knopov P. S. Control of spatially structured random processes and random fields with applications. Boston/Dordrecht/London, 2006.
3. Gubenko L. G., Shtatland Controlled Markov and semi-Markov models and some specific problems of optimization of stochastic systems. 1973.
4. Implement Value Iteration in Python [Електронний ресурс] –. - Режим доступу до ресурсу: https://www.geeksforgeeks.org/implement-value-iteration-in-python/.
5. Puterman Martin L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, 2005.
6. Ross Sheldon Introduction to Stochastic Dynamic Programming. Academic Press. 1983.

ДОДАТОК А

```
import itertools
import numpy as np


def read_parameters():
    l = float(input("Введіть λ (інтенсивність надходження заявок): "))
    c = int(input("Введіть кількість сервісів c: "))
    K = int(input("Введіть довжину черги K: "))

    mu = []
    C_i = []
    S_i = []
    for i in range(c):
        mu.append(float(input(f"Введіть μ_{i + 1} (інтенсивність обслуговування сервісу {i + 1}): ")))
        C_i.append(float(input(f"Введіть C_{i + 1} (витрати на утримання сервісу {i + 1}): ")))
        S_i.append(float(input(f"Введіть S_{i + 1} (витрати на активацію сервісу {i + 1}): ")))

    C_wait = float(input("Введіть C_wait (витрати за очікування у черзі): "))
    M = int(input("Введіть кількість типів заявок: "))

    R_j = []
    p_j = []
    for j in range(M):
        R_j.append(float(input(f"Введіть R_{j + 1} (дохід від заявки типу {j + 1}): ")))
```

```
        p_j.append(float(input(f"Введіть p_{j + 1} (ймовірність заявки типу {j + 1}):
"")))

    return l, c, K, mu, C_i, S_i, C_wait, M, R_j, p_j



def generate_states(c, K):
    states = []

    for Q in range(K + 1):
        for D in itertools.product([(0, 0), (1, 0), (1, 1)], repeat=c):
            if Q > 0 and all(active == 0 for (active, busy) in D):
                continue

            states.append((Q, D))

    return states



def generate_actions(s, K):
    Q, D = s
    c = len(D)
    acts = []

    for u in itertools.product([0, 1], repeat=c):
        if all(u_i >= busy for (active, busy), u_i in zip(D, u)):
            if sum(u) == 0:
                continue

            if Q < K:
```

```
            acts.append((u, 'accept'))
        else:
            acts.append((u, 'reject'))

    return acts


def reward(s, a, C_i, S_i, C_wait, R_j, p_j):
    Q, D = s
    u, decision = a

    revenue_flow = sum(p * R for p, R in zip(p_j, R_j))
    running_costs = sum(C for (active,_), C in zip(D, C_i) if active)
    waiting_cost = Q * C_wait

    I_s = [i for i, ((active,_), u_i) in enumerate(zip(D, u)) if active == 0 and u_i == 1]
    switch_cost = sum(S_i[i] for i in I_s)

    return revenue_flow - running_costs - waiting_cost - switch_cost


def is_valid_state(Q, D):
    return not (Q > 0 and all(active == 0 for (active, busy) in D))


def filling_services(Q, D):
    D = list(D)

    for i in range(len(D)):
        active, busy = D[i]
```

```
        if active == 1 and busy == 0 and Q > 0:
            D[i] = (1, 1)
            Q -= 1

    return Q, tuple(D)



def probabilities_sa(s, a, l, mu):
    Q, D = s
    u, decision = a

    D_after = tuple((u_i, busy) for (active,busy), u_i in zip(D, u))
    Q_after_2, D_after_2 = filling_services(Q, D_after)
    delta = l + sum(mu_i for (active,busy), mu_i in zip(D_after_2, mu) if busy)

    probs = {}
    prob_arr = l / delta
    Q_new = Q_after_2

    if decision == 'accept':
        Q_new = Q_after_2 + 1

    if is_valid_state(Q_new, D_after_2):
        probs[(Q_new, D_after)] = probs.get((Q_new, D_after_2), 0) + prob_arr

    for i, ((active,busy), mu_i) in enumerate(zip(D_after_2, mu)):
        if busy:
            prob_dep = mu_i / delta
            D_dep = list(D_after_2)
```

```
            D_dep[i] = (active, 0)

            if Q_after_2 > 0:
                D_dep[i] = (active, 1)
                Q_dep = Q_after_2 - 1
            else:
                Q_dep = Q_after_2

            D_dep = tuple(D_dep)

            if is_valid_state(Q_dep, D_dep):
                probs[(Q_dep, D_dep)] = probs.get((Q_dep, D_dep), 0) + prob_dep

    return probs, delta


def value_iteration(s, a_map, l, mu, C_i, S_i, C_wait, R_j, p_j, eps=1e-6,
max_iter=10000):
    V = {i: 0.0 for i in s}
    g = 0.0
    s0 = s[0]

    for it in range(max_iter):
        V_old = V.copy()

        for i in s:
            best = -np.inf

            for a in a_map[i]:
                R_sa = reward(i, a, C_i, S_i, C_wait, R_j, p_j)
```

```
            probs, delta = probabilities_sa(i, a, l, mu)

            expect = sum(p * V_old[s2] for s2, p in probs.items())
            val = R_sa / delta + expect - g / delta
            best = max(best, val)

        V[i] = best

    g += V[s0] - V_old[s0]
    if max(abs(V[i] - V_old[i]) for i in s) < eps:
        break

policy = {}

for i in s:
    best_actions = []
    best_val = -np.inf

    for a in a_map[i]:
        R_sa = reward(i, a, C_i, S_i, C_wait, R_j, p_j)
        probs, delta = probabilities_sa(i, a, l, mu)
        expect = sum(p * V[s2] for s2, p in probs.items())
        val = R_sa / delta + expect

        if val > best_val + 1e-9:
            best_val = val
            best_actions = [a]

        elif abs(val - best_val) < 1e-9:
            best_actions.append(a)
```

```
        policy[i] = best_actions

    return policy, g


def main():
    l, c, K, mu, C_i, S_i, C_wait, M, R_j, p_j = read_parameters()

    states = generate_states(c, K)
    actions_map = {s: generate_actions(s, K) for s in states}

    policy, g = value_iteration(states, actions_map, l, mu, C_i, S_i, C_wait, R_j, p_j)

    print("\nОптимальна стратегія:")

    for s in states:
        acts = policy[s]

        for a in acts:
            u, decision = a
            print(f" Стан (Q={s[0]}, D={s[1]}) → u={u}, рішення: {('прийняти заявку' if decision == 'accept' else 'відхилити заявку')}")

    print(f"\nСередній максимальний дохід за одиницю часу: {g:.2f}")


if __name__ == "__main__":
    main()
```