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

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«ЛОКАЛЬНЕ КЕРУВАННЯ В СТОХАСТИЧНИХ РОЗПОДІЛЕНИХ СИСТЕМАХ»**

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

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

Рецензент _____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК_____
(підпис)

«_____» _____ 20__р.

Київ - 2022

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

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
проф., д.ф.-м.н.
_____ Б. В. Олійник
(підпис)
"____"_____ 20__ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на класифікаційну роботу
студенту 4-го року навчання факультету інформатики
Катричу Костянтину Вячеславовичу

Тема: Локальне керування в стохастичних розподілених системах.

Зміст ТЧ до класифікаційної роботи:

Індивідуальне завдання

Вступ

1. Стохастичні системи

2. Керування в стохастичних системах з локальною взаємодією

3. Задача локального керування в стохастичних розподілених системах

Висновок

Література

Додаток

Дата видачі "____"_____ 2021 Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: Локальне керування в стохастичних розподілених системах.

Календарний план виконання роботи:

№	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	21.10.2021	
2.	Огляд технічної літератури за темою роботи.	21.03.2022	
3.	Аналіз методу керування Дермана.	21.04.2022	
4.	Розробка примітивної програми для роботи.	10.04.2022	
5.	Аналіз та удосконалення програми.	01.05.2022	
6.	Написання теоретичної частини роботи.	10.06.2022	
7.	Аналіз отриманих результатів з керівником та передзахист.	14.06.2022	
8.	Корегування роботи за результатами попереднього захисту.	24.06.2022	
9.	Остаточне оформлення роботи.	27.06.2022	
10.	Захист кваліфікаційної роботи.	04.07.2022	

Студент **Катрич К. В.**_____
(підпис)

Керівник к.ф.-м.н. **Чорней Р. К.**_____
(підпис)

Зміст

Анотація

В роботі досліджуються стохастичні розподілені системи та локальне керування в них. Загалом розглядається керування в стохастичних моделях для дискретного часу, а саме Ланцюги Маркова. В теоретичній частині роботи водяться загальні означення та теоретичні факти про стохастичні розподілені системи. Описуються означення та теореми, які спростять керування випадковими процесами. Розглядається та застосовується метод С. Дермана для покращення стратегії локального керування. В роботі наводиться приклад розв'язання задач за цим методом, завдяки власній програмі. Розглянуті проблеми з якими можна зіткнутися при розробці цієї програми та у додатку наведено сам програмний код мовою Python.

Ключові слова: Ланцюг Маркова, керований випадковий процес, локальне керування, стохастичні розподілені системи.

Вступ

Актуальність цієї теми обумовлена широким її використанням в інших сферах та дослідженнях. А саме для аналізу й управління різних апаратів, моделей масового обслуговування, прогнозування потоків трафіку, комунікаційних мереж, генетичних проблем та інших. Так, наприклад, гру "змійки та драбини"можна представити, як ланцюг Маркова. Стан системи - це дошка на якій йде гра, зі всіма гравцями. Та наш перехід до іншого стану повністю випадковий, бо залежить лише від кидка гральних кубиків. Розробити фізичну модель для цих хаотичних систем було б неймовірно складно, але зробити це за допомогою ланцюгів Маркова досить просто. Також, інші складніші моделі, які вираховують значення в стохастичних системах у своїй основі, використовують такі ланцюги чи їх видозміну версію. Доволі довгий час Google використовував метод обчислення рейтингу вебсторінок, в основі якого лежить саме ланцюгом Маркова.

Об'єкт дослідження моєї роботи це стохастичні розподілені системи. Предметом дослідження стало локальне керування в цих системах та їх поведінка відповідно до певних стохастичних правил. Загалом розглядається керування в стохастичні моделі для дискретного часу, а саме Марківські ланцюги. Ланцюг Маркова — це дискретна послідовність станів, кожен із яких береться з дискретного простору станів (кінцевого чи нескінченного), що задовольняє марковській властивості, тобто ймовірність кожної події залежить від стану, досягнутого тільки у попередній події. При локальному керуванні в таких системах ми можемо обирати певну стратегію, яка впливає на подальшу роботу системи, конкретно для кожного з можливих станів цієї системи. При цьому стратегія обирається лише на основі обмеженої інформації.

Мета дослідження - розробка методу покращення стратегії локального керування в стохастичних розподілених системах та створення програми для автоматичного знаходження оптимальної стратегії, базуючись на дослідженнях.

Методи дослідження - теорія ймовірностей та математична статистика. Для практичної роботи на мові Python використовувалась інтегроване середовище розробки PyCharm.

Робота поділяється на практичну та теоретичну частину. В першому розділі водяться загальні означення та теоретичні факти, які потрібні для подальшої роботи. В другому розділі описуються означення та теореми, які спростять керування такою системою. В третьому розділі наведено абстрактний приклад задачі у сфері обслуговування та розв'язання цього прикладу шляхом використання власної програми для автоматичного пошуку оптимальної стратегії.

1 Стохастичні системи

Стохастичні системи - це системи, зміни в яких відбуваються під впливом випадкових факторів. Для їх опису вводиться випадковий оператор, який формує простір елементарних подій з ймовірнісною мірою.

Стохастичний процес — це процес, який складається з випадкових змінних, значення яких змінюються в часі $X(t)$ або X_t де $t \in \mathbb{N}$ визначених на загальному ймовірнісному просторі $(\Omega, \mathcal{F}, P)$ де Ω - це зразок простору, $\mathcal{F}$-це σ-алгебра, а P - ймовірнісна міра.

Як приклад можна навести процес Бернуллі. Процес Бернуллі - це кінцева або нескінченна послідовність двійкових випадкових змінних 0 та 1 з ймовірністю p та $q \equiv 1 - p$ відповідно. Випадковий результат підкидання монетки як раз і є прикладом процесу Бернуллі.

1.1 Ланцюг Маркова

Марковський процес - це стохастичний процес, що задовольняє Марківській властивості, тобто, умовний розподіл ймовірностей майбутніх станів процесу залежить тільки від нинішнього стану, а не від послідовності подій, які передували цьому. [4]

Означення 1.1. Випадкова величина ξ визначена на $(\Omega, \mathcal{F}, P)$ володіє Марківською властивістю, якщо виконується наступне [5]:

$$\mathbb{P}(\xi_n = x_n \mid X_{n-1} = x_{n-1}, \ldots, \xi_0 = x_0) = P(X_n = x_n \mid \xi_{n-1} = x_{n-1}) \qquad (1)$$

Марковський процес може бути з дискретним та неперервним часом. У Марківському процесі з неперервним часом ми також маємо дискретний набір станів, як й процесі з дискретним часом. Однак поведінка переходу відрізняється: у кожному стані є ряд можливих події, які можуть викликати перехід. Подія, яка викликає перехід відбувається через експоненційний проміжок часу, тобто в цій моделі переходи відбуваються у випадкові моменти часу.

Ланцюг Маркова - це Марківський процес з дискретним часом і дискретним простором станів. Отже, ланцюг Маркова — це дискретна послідовність станів, кожен із яких береться з дискретного простору станів (кінцевого чи нескінченного).

Означення 1.2. Ланцюгом Маркова з дискретним часом називається послідовність дискретних випадкових величин $\xi = \{\xi^t, t = 0, 1, \dots\}$ з Марківською властивістю 1 [6]

Через те, що структура наших систем визначається взаємодією графів, то вводимо необхідні передумови та позначення. Нехай $G = (V, E)$ - кінцевий орієнтований граф з множиною вершин V та множиною ребер B. Позначимо k, j як з'єднані вершини графа k та j. Окіл вершини k є множина вершин сполучена ребрами з вершиною k: $N(k) = \{j : \{k, j\} \in E\}$. Повний окіл вершини k називається окіл вершини k разом з цією вершиною: $\widetilde{N}(k) = N(k) \cup \{k\}$

Нехай $X_i = \{x_i^1, \dots, x_i^{n_i}\}$ - локальний простір стану системи вузлів $i \in V$, тобто існує скінченна множина допустимих значень, пов'язаних з вершиною i. Тоді $\times_{i \in V} X_i$ - глобальний простір всіх станів системи. Для кожної підмножини вершин $K \subset V$ вектор $x_k := (x_k, k \in K) \in X_k = \times_{i \in V} X_i$ позначає опис граничного стану у вузлах K. Для всіх випадкових величин ми фіксуємо ймовірнісний простір $(\Omega, \mathcal{F}, P)$.

Означення 1.3. $P(n)$ де $P_{ij}(n) \equiv \mathbb{P}(x_{n+1} = j \mid x_n = i)$ називається матрицею ймовірностей переходу на n-му кроці.
Така матриця стохастична, тобто $\sum\limits_{j=1}^{\infty} P_{ij}(n) = 1, \quad \forall n \in \mathbb{N}.$

Для практичної роботи нам потрібні будуть саме ланцюги Маркова з локальною взаємодією, тому спершу треба дати означення для них.

1.2 Ланцюги Маркова з локальною взаємодією

Означення 1.4. Ланцюги Маркова з локальною взаємодією називаються такі ланцюги Маркова, при яких перехідні ймовірності задовольняють наступну умову

$$\mathbb{P}\{\xi_k^{t+1} \in C_k \mid \xi^t = x^t, \dots, \xi^0 = x^0\} = \mathbb{P}\{\xi_k^{t+1} \in C_k \mid \xi^t_{\widetilde{N}(k)} = x^t_{\widetilde{N}(k)}\} \quad (2)$$

де $k \in V, x^0, \dots, x^t \in X, C_k \in \mathcal{X}$
Тобто імовірність переходу у вершині k залежить від стану його повного околу у попередньому моменті часу. [7]

Означення 1.5. Перехідні ймовірності з ланцюга Маркова є синхронними, якщо

$$\mathbb{P}(\xi^{t+1} = x^{t+1} \mid \xi^t = x^t) = \prod_{k \in V} \mathbb{P}(\xi_k^{t+1} = x_k^{t+1} \mid \xi^t = x^t) \quad (3)$$

де $K \subset V, x^t, x^{t+1} \in X$

Означення 1.6. Для будь-якого часу $t \in \mathbb{N}$

$$\mathbb{P}(\xi^{t+1} = x^{t+1} \mid \xi^t = x^t) = \prod_{k \in V} \mathbb{P}(\xi_k^{t+1} = x_k^{t+1} \mid \xi^t_{\widetilde{N}(k)} = x^t_{\widetilde{N}(k)}) \quad (4)$$

де $K \subset V, x^t, x^{t+1} \in X$

Якщо ξ виконує умови 2 і 3, він називається Марківським процесом з локально взаємодіючими синхронними компонентами над простором (G, X).

2 Керування в стохастичних системах з локальною взаємодією

В цій роботі ми будемо розглядати керування в стохастичних системах з локальною взаємодією у сфері обслуговування черг, де вершини будуть незалежними серверами Бернуллі при FCFS (First-Come-First-Served). Якщо в час t на вершині j був обслужений користувач, та на цій вершині кількість користувачів $h-1 \geq 0$, то з ймовірністю $p_{ji}(h) \in (0,1)$ цей користувач перейде до іншої вершини i в момент часу $(t+1)$, або залишиться в цій вершині з ймовірністю $q_j = 1 - \sum_{i \in V} p_{ji}(h)$. Якщо в якійсь вершині в один й той самий час клієнт потрапляє до вершини, а інший клієнт залишає її, то вважаємо, що подія залишення вершини відбувається перед подією потрапляння в вершину. [3] Ці умови як раз задовольняють умови локальності 2, та є синхронними відповідно до 3.

Керування стохастичною системою з Марківською властивістю доволі складне. Отримуємо ми інформацію тільки з околу вершини, а через функцію втрат складно сказати, яка стратегія буде оптимальною. Для того, щоб полегшити розв'язання задачі, введемо допоміжні означення та теореми.

Означення 2.1. (1) Загальний набір можливих стратегій (контрольних значень) це $U = \times_{i \in V} U_i$ над графом G, де U_i це множина можливих рішень (дій) для особи яка приймає рішення у вершині i. U_i є кінцевим та незалежним від часу.

(2) Загальна необмежена історія залежних рішень у момент часу t є
$\Delta^t = \delta^t(x_0, \ldots, x^t) = \{\Delta_i^t(x_0, \ldots, x^t) : i \in V\} \in \times_{i \in V} U_i, t = 0, 1, \ldots$

(3) Неприпустимість дій, залежних від часу і стану, вводиться шляхом фіксації для будь-якої вершини $i \in V$ та для будь-яких станів $x \in X$ й множин $U_i^t(x)$ допустимих дій для часу Δ_i^t якщо $\xi^t = x$

Означення 2.2. Нехай необмежена керуюча множина задана як в 2.1, тоді (1) Якщо в будь-який момент часу $t = 0, 1, \ldots$ рішення Δ_i^t для вузла i зроблено на основі історії повного околу $\widetilde{N}(i)$ тільки для і, тобто для $x^0_{\widetilde{N}(i)}, \ldots, x^t_{\widetilde{N}(i)}$, послідовність рішень $\delta_i = \{\Delta_i^t, t \in \mathbf{N}\}$ називається локальною стратегією для вершини і.

Таким чином, ми маємо незалежну від історії послідовність рішень $\Delta_i^t(x^0, \ldots, x^t) = \Delta_i^t(x^0_{\widetilde{N}(i)}, \ldots, x^t_{\widetilde{N}(i)})$. Ці функції визначені на просторі $X^{t+1}_{\widetilde{N}(i)} = \times_0^{t+1} X_{\widetilde{N}(i)}$ та значення взяті з U_i

(2) Локальна стратегія $\delta = (\delta_i : i \in V)$ задається функцією $\delta_i = \{\Delta_i^t, t \geq 0\}$ де

$$\Delta_i^0 = \delta_i^0(x^0_{\widetilde{N}(i)}), \ldots, \Delta_i^t(x^0_{\widetilde{N}(i)}, \ldots, x^t_{\widetilde{N}(i)}), \ldots,$$

послідовність рішень залежить від локальної історії для вузла і.

Означення 2.3. Локальна стратегія $\delta = (\delta : i \in V)$ називається локальною Марківською стратегією, якщо для всіх $x^0, x^1, \ldots, x^t \in X$ виконується $\Delta_i^t(x^0_{\widetilde{N}(i)}, \ldots, x^t_{\widetilde{N}(i)}) = \Delta_i^t(x^t_{\widetilde{N}(i)}), i \in V$, тобто кожна локальна функція рішень Δ_i^t залежить від усієї історії тільки теперішнього локального стану.

Безпосереднім наслідком цього є те, що Марківська стратегія δ, яка використовується як стратегія управління Марківським випадковим полем, залежить від часу і визначається Марківським процесом (ξ, δ).

Означення 2.4. Локальна Марківська стратегія $\delta = (\delta_i : i \in V)$ називається стаціонарною локальною Марківською стратегією, якщо $\Delta_i^{t'}(x_{\widetilde{N}(i)}) \equiv \Delta_i^{t''}(x_{\widetilde{N}(i)}), i \in V$ для всіх t' та t'', тобто стаціонарна Марківська стратегія δ повністю визначається функцією Δ_i з

$$\Delta_i : X_{\widetilde{N}(i)} \to U_i, i \in V;$$
$$\mathrm{x}^t_{\widetilde{N}(i)} \to \Delta_i(x^t_{\widetilde{N}(i)})$$

Локальні структури допустимих рішень для осіб, які контролюють свої локальні частини системи, повинні залежати від часу та від фактичного локального стану системи.

Означення 2.5. Пара послідовностей (ξ, δ) називається керованим процесом з локальною взаємодією синхронних компонентів, відносно остаточного графа $G = (V, E)$ де:
$\xi = (\xi^t, t \in \mathbf{N})$ - це стохастичний процес з простором станів $X = \times_{i \in V} X_i$.
$\delta = (\delta_i : i \in V)$ - це локальна стратегія.
Переходи визначені наступним чином:

$$
\begin{aligned}
&P\{\xi_S^{t+1} = x_s / \xi^0, \Delta^0(\xi^0) = u^0, \ldots, \xi^{t-1} = x^{t-1}, \\
&\Delta^{t-1}(\xi^0, \ldots, \xi^{t-1}) = u^{t-1}, \xi^t = y, \Delta^t(\xi^0, \ldots, \xi^t) = u\} \\
&\overset{(1)}{=} P\{\xi_S^{t+1} = x_S / \xi^0 = x^0, \ldots, \xi^{t-1} = x^{t-1}, \xi^t = y, \Delta^t(\xi^0, \ldots, \xi^t) = u\} \\
&\overset{(2)}{=} P\{\xi_S^{t+1} = x_S / \xi^t = y, \Delta^t(\xi^0, \ldots, \xi^t) = u\} \\
&\overset{(3)}{=} \prod_{j \in S} P\{\xi_j^{t+1} = x_j / \xi^t = y, \Delta^t(\xi^0, \ldots, \xi^t) = u\} \\
&\overset{(4)}{=} \prod_{j \in S} P\{\xi_j^{t+1} = x_j / \xi^t_{\widetilde{N}(j)} = y_{\widetilde{N}(j)}, \Delta^t(\xi^0_{\widetilde{N}(j)}, \ldots, \xi^t_{\widetilde{N}(j)}) = u_j\} \\
&\overset{(5)}{=} \prod_{j \in S} Q_j\{x_j / y_{\widetilde{N}(j)}, u_j\} \cdot 1_{\{u\}}(\Delta^t(\xi^0, \ldots, \xi^t)) \\
&\overset{(6)}{=} Q_S(x_S / y, u) \cdot 1_{\{u\}}(\Delta^t(\xi^0, \ldots, \xi^t)), S \subseteq V, y \in X, u \in U(y)
\end{aligned}
$$

де $Q_j\{x_j / y_{\widetilde{N}(j)}, u_j\}$ це локально визначені ядра переходу, які визначають незмінні в часі закони руху системи для $S = V : Q(x/y, u) = P\{\xi^{t+1} = x / \xi^t = y, \Delta^t = u\}$

Кожен перехід зумовлено означенням, яке було вже наведено в роботі. Перший вираз і є визначенням керованого стохастичного процесу. $\overset{(1)}{=}$ зумовлено структурою марківського переходу та ітеративним визначення правил прийняття рішень. $\overset{(2)}{=}$ є частиною припущення про марківський закон переходу. $\overset{(3)}{=}$ обумовлено синхронною дією компонентів. $\overset{(4)}{=}$ можливий завдяки локальності переходів. Незалежність ймовірностей однокрокового переходу від часу дозволяє зробити перехід $\overset{(5)}{=}$. Та $\overset{(6)}{=}$ зумовлено також синхронною дією координат.

Система, яка в момент часу $t \in \mathbf{N}$ знаходиться в стані $\xi^t = x^t$ та приймає рішення u^t, несе однокрокові витрати $r(x^t, u^t) \geq 0$. Середні очікувані витрати при стратегії δ в момент часу T при $\xi^0 = y$ визначаються наступним

чином:

$$Q_T^\delta(y) = E_y^\delta \frac{1}{T+1} \sum_{t=0}^{T} r(\xi^t, \Delta^t) := E_y^\delta \frac{1}{T+1} \sum_{t=0}^{T} r(\xi^t, \Delta^t(\xi^0, \ldots, \xi^t))$$

де E_y^δ - це очікування, пов'язане з контрольованим процесом (ξ, δ).

Головною задачею керування в стохастичних розподілених системах полягає в тому, щоб знайти оптимальну стратегію, при якій мінімізуються середні очікувані витрати R^δ

$$R_y^\delta = \limsup_{T \to \infty} Q_T^\delta(y)$$

Позначимо це як

$$\rho(y) = inf_{\delta \in LD} R_y^\delta$$

Означення 2.6. Стратегія δ' називається оптимальною, якщо для всіх $y \in X$ виконується $\rho(y) = R_y^{\delta'}$

Теорема 2.1. Розглянемо керовані процеси (ξ, δ) з локальною взаємодією синхронних компонентів відносно графа $G = (V, E)$ згідно з Означенням 2.5 зі скінченим простором станів X та скінченим простором рішень U. Нехай множина допустимих дій $U_t(\cdot)$ не залежить від часу t. Тоді в класі LD допустимих детермінованих локальних стратегій існує єдина оптимальна стратегія, яка знаходиться в класі LD стаціонарних Марківських стратегій.

З доведенням теореми можна ознайомитись в роботі [1]

3 Задача локального керування в стохастичних розподілених системах

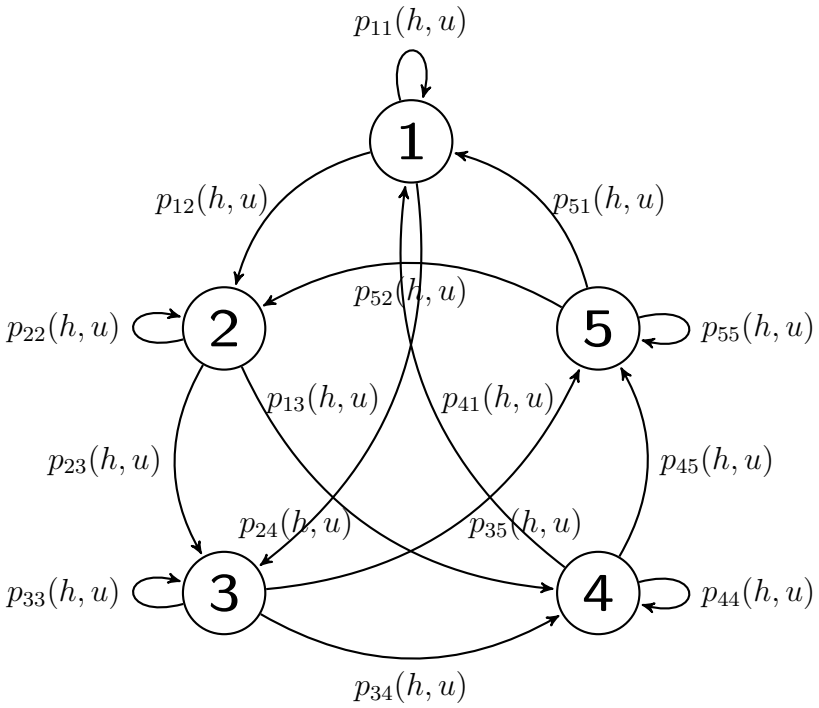
В цьому розділи описується постановка та розв'язання практичної задачі для якої виводились всі означення та теореми.

3.1 Модель задачі

Нехай ми маємо 5-ть вузлів $V = 5$ та 3-х клієнтів $K = 3$. Кожен клієнт після того, як вузол j завершив обслуговування, може залишитися в цьому самому вузлі, перейти в наступний вузол або перейти в вузол через один з ймовірностями $q_j(h,u), p_{j,j+1}(h,u)$ та $p_{j,j+2}(h,u)$, де h - це кількість клієнтів в цьому вузлі, u - рішення в цьому вузлі. В першому, третьому та п'ятому вузлі ми можемо обирати одне з двох рішень $0, 1$, а в другому та четвертому вузлі - одне з трьох рішень $0, 1, 2$.

Вибір ймовірностей обслуговування здійснюється за бінарними альтернативами з $U_j = 0, 1$ для $j \in 1, 3, 5$ та $U_j = 0, 1, 2$ для $j \in 2, 4$

Граф буде виглядати так:



Наприклад, якщо задамо вектори переходу з кожного в кожен вузол $p_{ji}(h,u)$ де $i,j \in V, u \in U_j$:

$$\begin{pmatrix} ((p_{11}(1,0);p_{11}(1,1));(p_{11}(2,0);p_{11}(2,1));(p_{11}(3,0);p_{11}(3,1))) \\ ((p_{12}(1,0);p_{12}(1,1));(p_{12}(2,0);p_{12}(2,1));(p_{12}(3,0);p_{12}(3,1))) \\ ((p_{13}(1,0);p_{13}(1,1));(p_{13}(2,0);p_{13}(2,1));(p_{13}(3,0);p_{13}(3,1))) \\ ((p_{14}(1,0);p_{14}(1,1));(p_{14}(2,0);p_{14}(2,1));(p_{14}(3,0);p_{14}(3,1))) \\ ((p_{15}(1,0);p_{15}(1,1));(p_{15}(2,0);p_{15}(2,1));(p_{15}(3,0);p_{15}(3,1))) \end{pmatrix} =$$

$$\begin{pmatrix} ((0.2,0.25),(0.3,0.25),(0.35,0.4)) \\ ((0.6,0.65),(0.3,0.25),(0.1,0.15)) \\ ((0.2,0.1),(0.4,0.5),(0.55,0.45)) \\ 0 \\ 0 \end{pmatrix}$$

Та аналогічно для інших вузлів

$$p_{2i}(h,u), i \in V = \begin{pmatrix} 0 \\ ((0.75,0.85,0.9),(0.65,0.7,0.75),(0.55,0.6,0.65)) \\ ((0.15,0.10,0.05),(0.30,0.25,0.20),(0.05,0.05,0.05)) \\ ((0.10,0.05,0.05),(0.05,0.05,0.05),(0.4,0.35,0.30)) \\ 0 \end{pmatrix}$$

$$\mathrm{p}_{3i}(h,u), i \in V = \begin{pmatrix} 0 \\ ((0.75,0.85,0.9),(0.65,0.7,0.75),(0.55,0.6,0.65)) \\ ((0.15,0.10,0.05),(0.30,0.25,0.20),(0.05,0.05,0.05)) \\ ((0.10,0.05,0.05),(0.05,0.05,0.05),(0.4,0.35,0.30)) \\ 0 \end{pmatrix}$$

$$\mathrm{p}_{4i}(h,u), i \in V = \begin{pmatrix} 0 \\ 0 \\ ((0.05,0.30),(0.10,0.60),(0.20,0.55)) \\ ((0.05,0.40),(0.10,0.05),(0.05,0.1)) \\ ((0.9,0.3),(0.8,0.35),(0.75,0.35)) \end{pmatrix}$$

$$\mathrm{p}_{5i}(h,u), i \in V = \begin{pmatrix} ((0.1,0.15,0.20),(0.40,0.30,0.20),(0.30,0.50,0.70)) \\ 0 \\ 0 \\ ((0.1,0.15,0.20),(0.1,0.60,0.20),(0.20,0.10,0.05)) \\ ((0.8,0.7,0.60),(0.5,0.10,0.6),(0.5,0.4,0.25)) \end{pmatrix}$$

Система, що перебуває в момент часу t в стані ξ^t та керована рішенням u^t, буде зазнавати однакових втрат $r(x,u)$, які задамо функцією:

$$r(x,u) = \sum_{j \in S}[(c_j - u_j) * x_j + u_j * b_j]$$

де c = (1, 2, 3, 2, 1) - додаткові витрати за перебування в вершині, b = (3, 1, 2, 2, 1) - додаткові витрати за прийняття рішення в вершині. Саме ці витрати ми будемо намагатися мінімізувати в задачі.

Ця система загалом має 35 різних станів S(5,3)

$$x_1 = (0,0,0,0,3), x_2 = (0,0,0,1,2), x_3 = (0,0,0,2,1), x_4 = (0,0,1,1,1)$$
$$\dots$$
$$\mathrm{x}_{32} = (2,0,0,1,0), x_{33} = (2,0,1,0,0), x_{34} = (2,1,0,0,0), x_{35} = (3,0,0,0,0)$$

Та 72 різних рішень:

$$u_1 = (0,0,0,0,0), u_2 = (0,0,0,0,1), u_3 = (0,0,0,1,0), u_4 = (0,0,0,1,1)$$
$$\dots$$
$$\mathrm{u}_{69} = (1,2,1,1,0), u_{70} = (1,2,1,1,1), u_{71} = (1,2,1,2,0), u_{72} = (1,2,1,2,1)$$

Програма створює множину станів та рішень автоматично:

```
all_decision = construct_all_decision()
X_states = construct_all_states()
```

3.2 Циклічний крок покращення стратегії керування

Розглянемо циклічний метод покращення стратегії керування для Марківського процесу прийняття рішень, запропонований С. Дерманом [2].

Спочатку ми обираємо довільну стратегію δ^0 для того, щоб знайти перші середні очікувальні витрати, які потім будемо покращувати. Знаходимо ядра переходу для цієї стратегії $Q(x/y,\delta)$ та знаходимо стаціонарний розподіл системи при стратегії δ^0 - π^0 завдяки системі рівнянь:

$$\begin{cases} \pi^{KJ} = \pi^{KJ} Q \\ \sum \pi^{KJ} = 1 \end{cases}$$

Обчислюємо однокрокові витрати r^0 та середні очікувані витрати R^0.

Далі задача полягає в знаходженні значень $\upsilon = (\upsilon(x) : x \in X)$. Обчислити значення υ можна із системи рівнянь:

$$\begin{cases} R_y^\delta + \nu(y) = r(y,\delta(y)) + \sum_{xS(K,J)} Q(x/y,\delta(y))\nu(x), yS(K,J) \\ \sum_{xS(K,J)} \pi^\delta(x)\nu(x) = 0 \end{cases}$$

Далі ми порівнюємо результати, які ми знайшли, з альтернативними значеннями які могли б бути при інших рішеннях. Для кожного стану y ми визначаємо множину альтернативних рішень U^y при яких показники були б кращими за знайдені.

Для знаходження кращих рішень для кожного стану, використовуємо системою рівнянь:

$$\begin{cases} \sum_{x\in S(K,J)} Q(x/y,u) R_x^\delta \le R_y^\delta \\ r(y,u) + \sum_{x\in S(K,J)} Q(x/y,u)\nu^\delta(x) < r(y,\delta(y)) + \sum_{x\in S(K,J)} Q(x/y,\delta(y))\nu^\delta(x) \end{cases}$$

або якщо спростити

$$\begin{cases} \sum Q(x/y,u) R_x^\delta \le R_y^\delta \\ r(y,u) + \sum Q(x/y,u)\nu^\delta(x) < R_y^\delta + \nu^\delta(y) \end{cases} \tag{5}$$

Якщо множина $U^y = \emptyset$, то рішення для цього стану і є найкращим. Якщо ж множина $U^y \neq \emptyset$, тоді ми формуємо нову стратегію з довільних рішень

з множини U^y для кожного стану $y \in X$ та повторюємо цей циклічний крок доки $\forall y(U^y = \emptyset)$. На кожному кроці наші середні очікувані витрати будуть зменшуватись $R^{\delta'} \leq R^{\delta}$ доки не стануть мінімальними. В результаті стратегія δ' буде вважатись оптимальною.

3.3 Розв'язання задачі

Спочатку обчислюємо ядра переходу $Q(x/y, u)$ та всі однокрокові витрати для всього простору станів X та простору рішень U.

```
mapRewardsByStrategy = dict()
mapQByStrategy = dict()

for decision in all_decision:
    mapRewardsByStrategy[tuple(decision)] = calculate_rewards([decision
   for i in range(len(X_states))])
    mapQByStrategy[tuple(decision)] = constructQ([constructT(decision) for
    i in range(len(X_states))])
```

Будуємо два словники. В першому словнику ключ це u, а значення - крокові витрати при цьому рішені. В другому словнику ключ також u, а значення - ядра переходу при такому рішенні.

Починаємо обчислюємо циклічний крок покращення стратегії. Спочатку беремо довільну стратегію, припустимо $\delta = [u_1, u_1 \dots]$

```
delta = []
delta = createNewStrategy(delta)
```

Обчислюємо для цієї стратегії ядра переходу з вершини в вершину з урахуванням того, що для кожного стану маємо окреме рішення.

```
T = construct_all_T(delta)
```

Далі на основі матриці переходу з вершини в вершину будуємо ядра переходу з одного стану в інший $Q(x/y, \delta)$

```
Q = construct_Q(T)
```

$Q(x/y,\delta)$	x^1	x^2	x^3	x^4	x^5	...
x^1	0.05	0	0	0	0	
x^2	0.04	0.005	0	0	0	
x^3	0	0.05	0.01	0	0	
x^4	0	0	0.5	0.2	0	
x^5	0.045	0.0025	0	0	0.0025	
$\vdots$						$\ddots$

Рахуємо стаціонарний розподіл π^δ на основі ядер переходу, які отримали при цій стратегії

```
pi = calculate_stationary_distribution(Q)
```

	x^1	x^2	x^3	x^4	x^5	...
$\pi(x)$	0.0005582	0.001626	0.0002535	0.00002	0.004636	...

Далі рахуємо крокові витрати $r(\delta)$ для нашої стратегії

```
reward = calculate_rewards(delta)
```

	x^1	x^2	x^3	x^4	x^5	...
$r(x)$	3	4	5	6	5	...

Обчислюємо очікувальні середні витрати R^δ

```
R = calculate_average_expects_reward(pi, reward)
```

$$R^\delta = 5.36639645935938$$

Знаходимо υ^δ

```
upsilon = calculate_upsilon(Q, pi, reward)
```

	x^1	x^2	x^3	x^4	x^5	...
$\upsilon(x)$	-7.015	-5.91	-4.133	-2.806	-4.781	...

Створюємо множину альтернативних кращих рішень U^y для кожного стану системи

```
U_y = find_alternative_strategy(upsilon, R, delta)
```

Порівнюємо середні очікувальні витрати з альтернативними за системою 5

```
better_decisions = []
for decision in pull_of_optimal_decision[tuple(y_state)][1:]:
    alternative_R = (mapQByStrategy[tuple(decision)][state_i] * R).sum()
    alternative_rewards = mapRewardsByStrategy[tuple(decision)][state_i] +
    mapQByStrategy[tuple(decision)][state_i].dot(v)
    this_rewards = R + v[state_i]
    if alternative_R <= R and alternative_rewards < this_rewards:
        better_decisions.append(decision)
```

	x^1	x^2	x^3	x^4	x^5	...
U^y	$\{u_2, u_{14}\}$	u_2	$\{u_2, u_6\}$	$\{u_3, u_5, u_6, u_{17}\}$	u_2	...

Повторюємо цей цикл, поки множина альтернативних рішень для кожного стану не буде порожньою $U^y = \emptyset$

На кожному кроці можна спостерігати зменшення середніх очікуваних витрат $R^{\delta'} < R^{\delta}$

$$R^{\delta_1} = 5.36639645935938$$
$$R^{\delta_2} = 4.708908576846542$$
$$R^{\delta_3} = 3.8005488560955465$$
$$R^{\delta_4} = 3.6170851008004936$$

Результат.

В результаті всіх обчислювань ми отримуємо оптимальну стратегію δ'

$\delta'(x^1) = u^2; \delta'(x^2) = u^2; \delta'(x^3) = u^2; \delta'(x^4) = u^5; \delta'(x^5) = u^2;$
$\delta'(x^6) = u^2; \delta'(x^7) = u^5; \delta'(x^8) = u^2; \delta'(x^9) = u^1; \delta'(x^{10}) = u^1;$
$\delta'(x^{11}) = u^2; \delta'(x^{12}) = u^2; \delta'(x^{13}) = u^5; \delta'(x^{14}) = u^2; \delta'(x^{15}) = u^1;$
$\delta'(x^{16}) = u^1; \delta'(x^{17}) = u^{25}; \delta'(x^{18}) = u^{25}; \delta'(x^{19}) = u^{25}; \delta'(x^{20}) = u^{25};$
$\delta'(x^{21}) = u^2; \delta'(x^{22}) = u^2; \delta'(x^{23}) = u^5; \delta'(x^{24}) = u^2; \delta'(x^{25}) = u^1;$
$\delta'(x^{26}) = u^1; \delta'(x^{27}) = u^{14}; \delta'(x^{28}) = u^1; \delta'(x^{29}) = u^1; \delta'(x^{30}) = u^{25};$
$\delta'(x^{31}) = u^2; \delta'(x^{32}) = u^1; \delta'(x^{33}) = u^1; \delta'(x^{34}) = u^1; \delta'(x^{35}) = u^{37};$

Та мінімальні очікувані середні витрати:

$$R^{\delta\prime} = 3.6170851008004936$$

Як можна побачити з приклада, власна програма з автоматичного пошуку оптимальної стратегії для стохастичних розподілених систем працює коректно. Ця програма позбавляє людину досить великих і об'ємних обчислень, а також автоматизує всі потрібні дії. Ця програма може легко редагуватись для вирішення будь-яких інших схожих прикладів та автоматизує їх вирішення, що багаторазово прискорює цей процес та усуває можливість помилитися в простих обчисленнях.

3.4 Проблеми при створенні програми

При створенні програми для автоматичного знаходження оптимальної стратегії я зіткнувся з деякими проблемами.

Обрахувати ядра переходу з кожного стану в кожен інший можливий стан виявилося доволі нетривіальною задачею. Треба було створити загальний алгоритм для знаходження можливих наступних станів та ймовірностей переходу до них. Через те що кожна вершина працювала синхронно з іншими, обрахувати всі можливі варіанти було доволі важко. Рішенням став алгоритм ітераційного розгляду кожної окремої вершини, та створення "дерева"можливих варіантів паралельно з обрахуванням ймовірностей цих варіантів.

Другою проблемою стали математичні дії над числами з плавучою точкою у двійковій системі числення. У зв'язку з технічною частиною мови програмування, при математичних діях над числами з плавучою точкою іноді з'являлись "дивні значення". Наприклад при додаванні 0.1 + 0.2 дорівнювало 0.300000000004. На перший погляд, це може бути занадто незначною похибкою, але іноді велика кількість таких похибок впливала на кінцевий результат обчислень. Розв'язання цієї проблеми стало округлення деяких значень до 5-го знака після крапки.

Висновок

У цій роботі були досліджені стохастичні системи, Марківські ланцюги та локальне керування в стохастичних розподілених системах. Було застосовано метод С. Дермана [2] для покращення стратегії керування. Його приклад було продемонстровано для обчислення оптимальної стратегії. Була розроблена та описана власна програма мовою Python для автоматичного пошуку оптимальної стратегії.

Література

[1] Ruslan K. Chornei, Hans Daduna, and Pavel S. Knopov, 2005 Controlled Markov Fields With Finite State Space on Graphs

[2] Derman, C. Finite State Markovian Decision Processes; Academic Press: New York, 1970.

[3] Gravey, A.; Hebuterne, G. Simultaneity in discrete-time single server queues with Bernoulli inputs. Performance Evaluation 1992, 14, 123–131.

[4] Гагнюк, Пол А. (2017). Ланцюги Маркова: від теорії до впровадження та експерименту. США, Нью-Джерсі: John Wiley and Sons. С. 1–235. ISBN 978-1-119-38755-8.

[5] Durrett, Rick. Probability: Theory and Examples. Fourth Edition. Cambridge University Press, 2010.

[6] Марков А. А., Распространение закона больших чисел на величины, зависящие друг от друга. — Известия физико-математического общества при Казанском университете. — 2-я серия. — Том 15. (1906) — С. 135—156.

[7] Vasilyev, N.B. Bernoulli and Markov stationary measures in discrete local interactions. In Locally Interacting Systems and Their Application in Biology; Dobrushin, R.L.; Kryukov, V.I.; Toom, A.L., Eds.; Lecture Notes in Mathematics 653; Springer: Berlin, 1978; 99–112.

Додаток

```
def construct_all_decision():
    arr = [[0 for i in range(nodeSet)]]
    u_i = 0
    for u in Uj:
        new_arr = []
        for k in arr:
            for d in u:
                new_k = copy.copy(k)
                new_k[u_i] = d
                new_arr.append(new_k)
        arr = new_arr
        u_i += 1

    return np.asarray(arr)


def construct_all_states():
    arr = [list(i) for i in product(tuple(i for i in range(K + 1)), repeat
   =nodeSet)]
    filtered = filter(lambda state_j: sum(state_j) == K, arr)
    return list(filtered)


all_decision = construct_all_decision()
X_states = construct_all_states()

pull_of_optimal_decision = dict()
for state in X_states:
    pull_of_optimal_decision[tuple(state)] = copy.deepcopy(all_decision)
```

Лістинг 1: Створення всіх станів та рішень

```
def calculate_rewards(u_j):
    rewards = []
    state_i = 0
    for x_j in X_states:
        rewards.append(calculate_reward(x_j, u_j))
        state_i += 1
    return np.array(rewards)


def constructT(u):
```

```
    T_j = []
    for k in range(nodeSet):
        T_k = []
        for j in range(len(u)):
            if T2[k][j] == 0:
                T_k.append(0)
            else:
                T_k.append([g2[u[k]] for g2 in T2[k][j]])
        T_j.append(T_k)
    return T_j


def canGoTo(T_delta, state, state_i):
    arr = [[copy.copy(state), 1]]
    p_i = 0
    for p in state:
        if p > 0:
            new_arr = []
            for i in arr:
                n = 0
                for w in T_delta[state_i][p_i]:
                    if w != 0:
                        new_position = copy.copy(i[0])
                        new_position[p_i] -= 1
                        new_position[n] += 1
                        new_arr.append([new_position, i[1] * w[p - 1]])
                    n += 1
            arr = new_arr
        p_i += 1
    return arr


def construct_Q(T_delta):
    Q = []
    state_i = 0
    for state in X_states:
        Q_j = np.zeros(len(X_states))
        canGoToArr = canGoTo(T_delta, state, state_i)
        canGoDict = {}
        for state_2 in canGoToArr:
            index = X_states.index(state_2[0])
            if index in canGoDict:
                canGoDict[index] += state_2[1]
```

```
            else:
                canGoDict[index] = state_2[1]
        for state_index in canGoDict:
            Q_j[state_index] = canGoDict[state_index]
        Q.append(Q_j)
        state_i += 1
    return Q


mapRewardsByStrategy = dict()
mapQByStrategy = dict()

for decision in all_decision:
    mapRewardsByStrategy[tuple(decision)] = calculate_rewards([decision
   for i in range(len(X_states))])
    mapQByStrategy[tuple(decision)] = construct_Q([constructT(decision)
   for i in range(len(X_states))])
```

Лістинг 2: Обрахунок всіх крокових втрат та ядер переходу

```
def construct_all_T(u_delta):
    T_delta = []
    for x_j in range(len(X_states)):
        T_delta.append(constructT(u_delta[x_j]))
    return T_delta


def calculate_stationary_distribution(Q):
    Q = np.array(Q)
    evals, evecs = np.linalg.eig(Q.T)
    evec1 = evecs[:, np.isclose(evals, 1)]
    evec2 = evec1[:, 0]
    stationary = evec2 / evec2.sum()
    stationary = stationary.real
    return stationary


def calculate_average_expects_reward(pi, r):
    return sum(pi[k] * r[k] for k in range(len(pi)))


def calculate_upsilon(Q, pi, reward):
    length = len(pi) + 1
    reward = np.append(reward, 0)
```

```
    auxiliary_matrix = np.subtract(np.identity(length - 1), Q)
    auxiliary_matrix = np.insert(auxiliary_matrix, 0, 1, axis=1)
    auxiliary_matrix = np.append(auxiliary_matrix, np.insert(pi, 0, 0))
    auxiliary_matrix = auxiliary_matrix.reshape(length, length)
    v = np.linalg.solve(auxiliary_matrix, reward)
    return np.array(v)[1:]



def find_alternative_strategy(v, R, u_delta):
    state_i = 0
    for y_state in X_states:
        better_decisions = []
        for decision in pull_of_optimal_decision[tuple(y_state)][1:]:
            alternative_R = (mapQByStrategy[tuple(decision)][state_i] * R)
    .sum()
            alternative_rewards = mapRewardsByStrategy[tuple(decision)][
    state_i] + mapQByStrategy[tuple(decision)][state_i].dot(v)
            this_rewards = R + v[state_i]
            if alternative_R <= R and alternative_rewards < this_rewards:
                better_decisions.append(decision)

        if len(better_decisions) == 0:
            better_decisions.append(u_delta[state_i])

        pull_of_optimal_decision[tuple(y_state)] = better_decisions
        state_i += 1
    return pull_of_optimal_decision


def create_new_strategy(u_delta):
    new_u = []
    for state_i in range(len(X_states)):
        if len(pull_of_optimal_decision[tuple(X_states[state_i])]) == 0:
            new_u.append(u_delta[state_i])
        else:
            new_u.append((pull_of_optimal_decision[tuple(X_states[state_i
    ])][0]))
    return new_u
```

Лістинг 3: Додаткові методи

```
if __name__ == '__main__':
    delta = []
```

```
    while not all((len(i) == 1) for i in pull_of_optimal_decision.values()
):
        delta = create_new_strategy(delta)
        T = construct_all_T(delta)
        Q = construct_Q(T)
        pi = calculate_stationary_distribution(Q)

        reward = calculate_rewards(delta)
        R = calculate_average_expects_reward(pi, reward)

        upsilon = calculate_upsilon(Q, pi, reward)

        U_y = find_alternative_strategy(upsilon, R, delta)

    for i in range(len(X_states)):
        print("Best decision for state {}: {}".format(X_states[i], delta[i
]))
```

Лістинг 4: Головний цикл