Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Курсова робота

освітній ступінь – бакалавр

на тему: **«МАРКОВСЬКІ МОДЕЛІ ПОВЕДІНКИ СИСТЕМ З КОНКУРУЮЧИМИ ТЕХНОЛОГІЯМИ»**

Виконав: студент 3-го року навчання,

Освітньої програми «Прикладна математика», 113

Хоптій Андрій Володимирович

Керівник: Чорней Р. К.,

доцент, кандидат фіз.-мат. наук

Київ – 2023

Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

ЗАТВЕРДЖУЮ
Зав. кафедри математики, проф.,
д.ф.-м.н.
Чорней Р. К.
_____
(підпис)
“____” _____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Хоптію А. В. факультету інформатики 3 курсу

ТЕМА «Марковські моделі поведінки систем з конкуруючими технологіями»

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1 Марковські моделі, їх типи і застосування
2 Поведінка систем з конкуруючими технологіями
3 Програмна реалізація моделі
Висновки
Список літератури
Додатки

Дата видачі “____”_____ 2023 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: «Марковські моделі поведінки систем з конкуруючими технологіями

Календарний план виконання роботи:

№	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу		
2.	Огляд наукової літератури за темою роботи		
3.	Детальний аналіз прочитаної літератури		
4.	Розробка програмної реалізації на основі розібраної літератури		
5.	Аналіз та перевірка роботи програми		
6.	Написання текстової частини курсової роботи		
7.	Аналіз результатів роботи над текстовою частиною та виправлення помилок		
8.	Створення слайдів для доповіді та написання доповіді		
9.	Захист курсової роботи		

Керівник Чорней Р. К.

“___”_____2023 р.

Зміст

Анотація

Робота “Марковські моделі поведінки систем з конкуруючими технологіями” присвячена загальному дослідженню Марковських моделей, їх практичному і теоретичному використанню на реальних прикладах. А також детальний розбір однієї з чотирьох моделей, а саме: “Марковський процес ухвалення рішень” та її подальша реалізація на прикладі систем з конкуруючими технологіями для максимізації прибутку.

Вступ

Конкуренція - це невід'ємна частина взаємовідносин, яка зустрічається всюди, інколи неспроможність конкурувати може призвести до фатальних наслідків. Якщо розглядати конкуренцію в сфері біології, то це боротьба за ресурси, які потрібні для життя: їжа, територія, особи протилежної статті, безпека тощо. Впродовж еволюції різни види живих організмів постійно шукають нові способи пристосування до подразників, якщо загрожують їх існуванню. У сфері бізнесу конкуренція доволі інтенсивна полягає у змаганні за найвигідніші умови збуду товарів споживачам, щоб отримати найбільший прибуток треба не тільки фокусуватись на якості і доступності свого продукту, а й постійно вивчати і прогнозувати можливі дії конкурентів, щоб виробляти нові оптимальні стратегії, на основі рішень, які вони приймають. Досконалий аналіз конкурентів може бути доволі комплексний і застосовувати такі сфери математики як теорія ймовірності, теорія ігор і статистичний аналіз.

Мета дослідження

Дати визначення, що таке марковські моделі, дослідити їх види і сфери застосування, розібратись як аналізувати і прогнозувати поведінку конкурентів у певних ситуаціях і знаходити оптимальні стратегії для прийняття найефективніших рішень боротьби з ними, щоб отримати найбільший прибуток

Актуальність дослідження

Як було зазначено вище, конкуренція є невід'ємною частиною нашого життя, від якої залежить не тільки наш дохід, а й виживання в сучасний реаліях. Актуальність дослідження конкуренції і розробка оптимальних стратегії розвитку завжди є актуальною темою. Зараз жоден бізнес не починає свою діяльність без глибокого аналізу ринку споживачів та конкурентів, які вже пропонують свої послуги в цій сфері. Аналіз ринку це не одноразова подія, треба постійно

слідкувати за багатьма навколишніми факторами і передбачати нові, щоб залишатись конкуренто спроможним.

У розділі 1 розписано основі поняття які стосуються марковських моделей, а також розглянуто по одному кожну з існуючих моделей, показано структури і застосування в реальному житті

У розділі 2 розписано про те, що таке конкуренція, до чого вона призводить, як ефективно складати конкуренцію іншим компаніям з застосування марковських процесів прийняття рішень в системі з конкуруючими технологіями.

У розділі 3 програмно змодельована система з конкуруючими технологіями та іде опис коду, які змінні за що відповідають, які методи використовуються, а також які бібліотеки/модулі потрібні для обчислень і виводу інформації на екран

Розділ 1. Марковські моделі, їх типи і застосування

1.1. Загальні поняття про Марковські моделі

Марковські моделі – це математична модель, для описання переходів від одного стану системи в інший, яка має властивість маркова.

Означення 1.1.1. Властивість називають Марковою, якщо наступний стан залежить тільки від теперішнього і ніяк не залежить від послідовності станів, які були до цього[1]:

$$P(\xi_{n+1} \mid \xi_0, \dots, \xi_{n-1}, \xi_n) = P(\xi_{n+1} \mid \xi_n)$$

Марковські моделі задаються скінченим числом станів, яка є в доступності у системи, а також матрицею переходів, де визначена ймовірність переходів від одного стану до іншого[2].

Пропоную розглянути приклад матриці переходів P, де x_{ij}, ймовірність переходу з одного стану в інший.

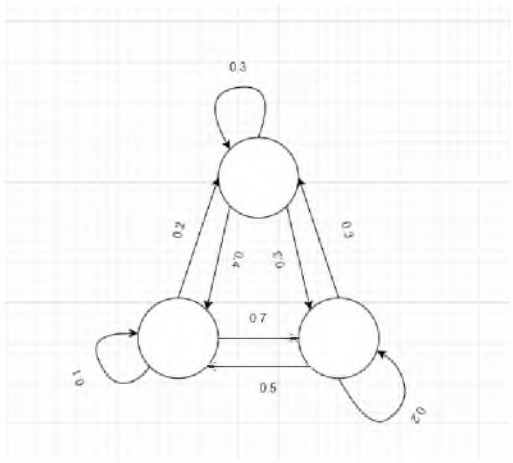
$$P = \begin{pmatrix} x_{1,1} & \cdots & x_{i,1} \\ \vdots & \ddots & \vdots \\ x_{1,j} & \cdots & x_{i,j} \end{pmatrix} \text{ де } i, j \in \text{N}$$

На практиці можна зустріти матриці переходів заданні різними способами.

Розглянемо нижче найпопулярніші:

0.1	0.6	0.3
0.45	0.2	0.35
0.5	0.3	0.2

Приклад 1. Матриця переходу задана таблицею



Приклад 2. Матриця переходів задана графом

```
474   const p = [
475     [0.3, 0.2, 0.5],
476     [0.4, 0.3, 0.3],
477     [0.1, 0.6, 0.3],
478   ];
```

Приклад 3. Матриця переходів задана двовимірним масивом в однопоточній, мультипарадигмальній мові програмування JavaScript

1.2. Типи Марковських моделей і їх застосування

Існує 4 типи загальних моделей, які відносяться до моделей Маркова:

- Ланцюг Маркова
- Прихована марковська модель
- Марковський процес ухвалення рішень
- частково спостережуваний марковський процес ухвалення рішень.

Всі ці моделі засовуються у різних ситуаціях і за різних умов. Далі пропоную розглянути кожну з цих моделей окремо.

1.2.1. Ланцюги маркова

Розглянемо серію випробувань, де $\xi_n, n \in \mathbb{Z}^+$ – це наслідок випробування, а n – порядковий номер самого випробування. **Ланцюгом Маркова** є послідовність $\{\xi_n, n \in \mathbb{Z}^+\}$, якщо результат випробування n буде залежати тільки від результату випробування n – 1 [1].

$$P(\xi_n = x_j | \xi_0 = x_{i_0}, \dots, \xi_{n-2} = x_{i_{k-1}}, \xi_{n-1} = x_i) = P(\xi_n = x_j | \xi_{n-1} = x_i)$$

Ланцюг Маркова використовується, коли у нас система автономна і у нас є можливість відслідковувати усі стани.

Однією з сфер використання ланцюгів маркова є біологія. Там вони застосовуються для опису популяційних процесів, у тому числі в **матриці Леслі**, яка відповідає не тільки за опис зростання популяції, а і за прогнозування вікового розподілу. У цій матриці розглядається популяція, яка не мігрує, розбита на групи за віком та знаходиться в необмеженому середовищі, а також розглядається тільки 1 стать, як правило жіноча[3][4].

Матрицю Леслі, яка складається з F – показника народжування, S – шанси на виживання для переходу з однієї вікової групи в іншу. Також ми маємо вектор популяції N_t і вектор результату N_{t+1}, який показує кількість осіб в кожній групі в році t+1

$$N_{t+1} = L \times N_t$$

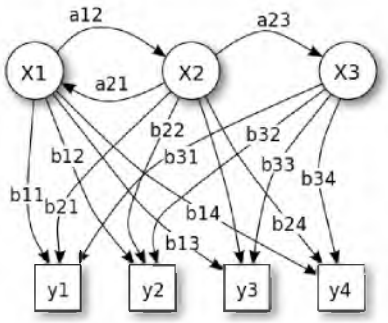
$$\begin{bmatrix} N_{0,t+1} \\ N_{1,t+1} \\ N_{2,t+1} \\ N_{3,t+1} \end{bmatrix} = \begin{bmatrix} F_0 & F_1 & F_2 & F_3 \\ S_0 & 0 & 0 & 0 \\ 0 & S_1 & 0 & 0 \\ 0 & 0 & S_2 & 0 \end{bmatrix} \times \begin{bmatrix} N_{0,t} \\ N_{1,t} \\ N_{2,t} \\ N_{3,t} \end{bmatrix}$$

1.2.2. Прихована марковська модель

Статична модель, яка імітує марковський процес з невідомими параметрами, для аналізу послідовності дискретних подій називається **прихованою марковською моделю**. На відміну від звичайної марковської моделі, яка була зазначена вище (***Означення 1.2.1.***), в даній структурі ми можемо спостерігати виключно за змінними, які впливають на поточний стан і не більше. Для кожного стану є свій розподіл ймовірності серед усіх можливих вихідних значень[5].

Застосовуються приховані марковські моделі дуже активно, їх можна часто зустріти в програмах, які використовуються для розпізнавання, рукописного введення, мелодій, акордів[6], мовлення[7][8], мови жестів тощо.

Структуру самої моделі добре можна зобразити на діаграмі. Позначимо момент часу символом t. Позначення x(t) буде випадковою прихованою змінною, яка має маркову властивість, тобто її значення залежить тільки від значення прихованої змінної x(t-1). Аналогічно маємо випадкову спостережувану змінну y(t), значення якої залежить напряму від значення x(t). Також потрібно позначити ймовірності переходів між станами і ймовірності виходів із станів, це будуть a_{ij} та b_{ij}.

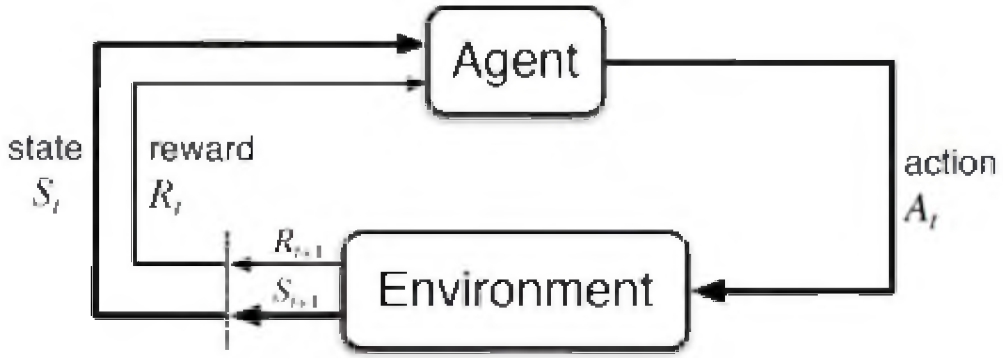


1.2.3. Марковський процес ухвалення рішень

Стохастичний процес ухвалення рішень, який використовує математично модель для моделювання прийняття рішень на динамічний системі, результат якої є або

випадковим або контролюється системою прийняття рішень, яка приймає послідовні рішення з часом, на основі поточного стану системи називається **Марковським процесом ухвалення рішень**. [9]

Структуру моделі зобразимо на діаграмі, де S_t – позначає стан системи на момент часу t, A_t – позначає прийняте рішення, $P(S_{t+1}| S_t, A_t)$ – матриця ймовірностей переходів з одного стану в інший, які залежить від прийнятого рішення та поточного стану системи, R_s – нагорода за прийняте рішення.

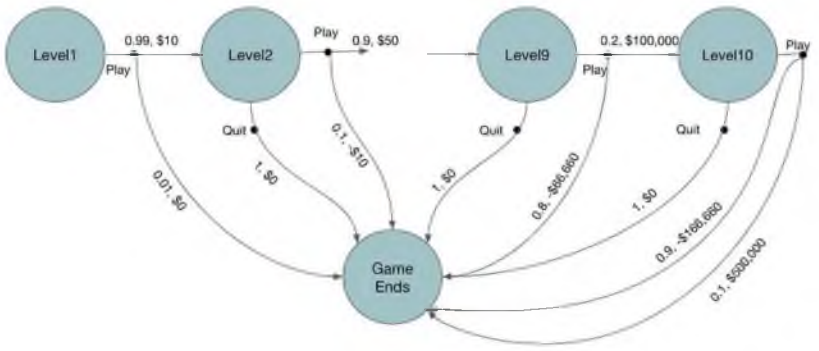


Модель має властивість маркова **(Означення 1.1.1)** , тому майбутній стан системи залежить лише від теперішнього стану, який зберігає всю інформацію всю необхідну інформацію минулих станів.

Задачі на Марковський процес ухвалення рішень застосовується активно для пошуку оптимальних стратегій дій для кожного можливого стану системи[10].

Такого типу задачі часто використовуються для зменшення затрат або для максимізації прибутку, що в більшості випадків є еквівалентно. Розглянемо це на прикладі гри в вікторину, де за кожне правильне питання ми отримуємо гроші, але якщо відповідь буде неправильно, то втратимо все. Звісно у нас на кожному рівні є

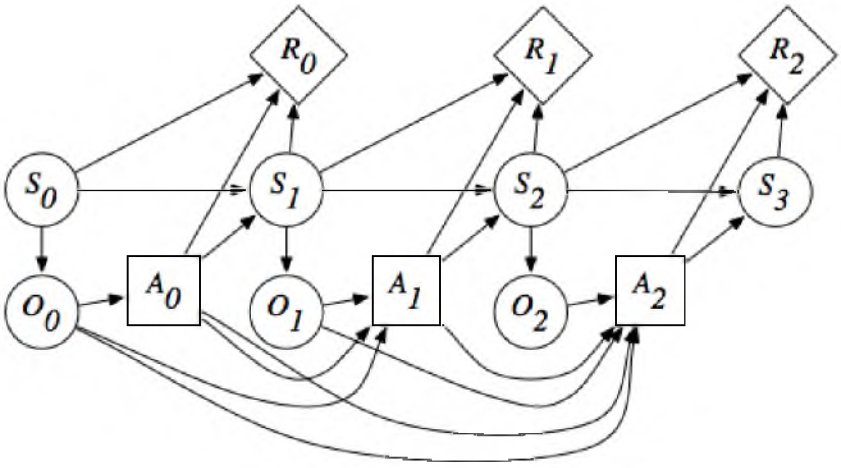
можливість забрати гроші і не продовжувати гру. Нижче приведено діаграму такої гри



1.2.4. Частково спостережуваний марковський процес ухвалення рішень

Модель, яка імітує прийняття рішень за умови невизначеності на основі марковських процесів з частковою спостережуваністю називається **Частково спостережуваний марковський процес ухвалення рішень**. [11]

Ця модель дуже схоже по структурі на Марковський процес ухвалення рішень, але ще додається декілька позначень. Давайте розглянемо це більш детально. Структура частково спостережуваного марковського процесу ухвалення рішень складається з S_t – стани системи, A_t – рішення системи, O_t – можливі спостереження, $P(S^* \mid S, A)$ – матриця переходів, яка буде показувати ймовірності переходів з одного стану в інший, $P(O \mid S)$ – матриця спостережень, яка буде показувати ймовірності отримання часткової інформації про стан системи, R_S [12]



Використати таку модель можна в покері [13], у нас є інформація про свої карти і про карти на столі, але немає інформації про те, які карти є у інших гравців, тому за допомогою частково спостережуваного марковського процесу ухвалення рішень, можна створити алгоритм прийняття рішень, який буде шукати найбільш ймовірний стан гри на основі спостережень, які є до цього моменту і на виході отримати інформацію чи варто нам продовжувати гру, піднямати ставки, робити чек або виходити з гри.

Розділ 2. Поведінка систем з конкуруючими технологіями

2.1. Основні положення конкуренції

Конкуренція, як правило, ведеться за ресурси, місця збуту, споживачів, а також за постачальників. Це все потрібно, щоб мінімізувати витрати і підвищити попит на свій продукт, що позитивно впливай на прибуток.

Конкуренція це позитивне явище, воно призводить до покращення якості продукту або послуг, зацікавлює зберігати продукт доступним і не підвищувати ціни на нього, в деяких випадків навіть знижувати, стимулює нові інновації, а також забезпечує вибір.

Щоб завжди залишатись конкурентоспроможним і знати свої слабкі місця, треба аналізувати не тільки свій продукт, а й всі навколишні фактори, такі як от конкурентів та споживачів. Ефективно це робити можна тільки за умови, що буде розглянуто наперед передбачувані сценарії розвитку конкурентів і на основі цього будуть вибудовані нові можливі стратегії розвитку і відібрати з них оптимальні, щоб щось протиставити конкурентам на ці дії.

2.2. Марковські моделі для прогнозування конкуренції та зростання кількості обчислень при змінні параметрів системи

Потужним інструментом прогнозування конкуренції між системами є марковські моделі поведінки систем з конкуруючими технологіями. Вони використовуються для пошуку найкращих стратегій керування системою, що призведе до максимального можливого прибутку. Модель відображає компанії, їх стани, їх рішення, а також безпосередньо зв’язок. Якщо компанія спостерігає за розвитком свого конкурента, то вона приймає рішення, не тільки на основі свого стану, а й враховує поточний стан конкурента. Це сильно збільшує можливі обрахунки. Пропоную розглянути приклад такої системи, щоб зрозуміти наскільки сильно зростають обрахунки.

Розглянемо систему компаній $N \in \{n1, n2, n3\}$, матрицю зв'язку між компаніями $L = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$. Кожна компанія буде мати свої рішення і свої стани. Для прикладу візьмемо ось такі стани: $S1 \in \{s1, s2, s3\}, S2 \in \{s1, s2, s3, s4\}, S3 \in \{s1, s2\}$ і ось такі рішення $A1 \in \{a1, a2\}$ $A2 \in \{a1, a2\}$ $A3 \in \{a1, a2\}$. Для рішень ще потрібно задати матрицю нагород. $R1 = (r1, r2), R2 = (r1, r2), R3 = (r1, r2)$. Також потрібна матриця переходів для кожної компанії. $P1 = (p1, p2, p3), P2 = (p1, p2, p3, p4), P3 = (p1, p2)$.

Тепер можна наглядно показати скільки можливих комбінацій станів і рішень існує при таких, на першу чергу, здавалось б несуттєвих параметрах.

Для компанії 1 будемо мати такі ймовірності переходу: $P\,(S1 \mid S1, S2, S3, A1)$, якщо обрахувати всі можливі комбінації, то отримаємо $3 * 3 * 4 * 2 * 2 = 144$ значень для кожної перестановки.

Для компанії 2 будемо мати такі ймовірності переходу: $P\,(S2 \mid S2, S1, A2)$, якщо обрахувати всі можливі комбінації, то отримаємо $4 * 4 * 3 * 2 = 96$ значень для кожної перестановки.

Для компанії 3 будемо мати такі ймовірності переходу: $P\,(S3 \mid S3, S1, A3)$, якщо обрахувати всі можливі комбінації, то отримаємо $2 * 2 * 3 * 2 = 24$ значень для кожної перестановки.

Ці ймовірності нам далі потрібні, щоб обрахувати Q – ймовірності переходів вже самої системи, а не окремо компаній.

$$Q\,(\,(S1, S2, S3) \mid (S1, S2, S3), (A1, A2, A3)\,) = P(S1 \mid S1, S2, S3, A1) * P(S2 \mid S2, S1, A2) * P(S3 \mid S3, S1, A3)$$

Комбінацій ймовірностей переходів з одного стану системи в інший буде $3 * 4 * 2 * 3 * 4 * 2 * 2 * 2 * 2 = 4608$

Якщо зменшити кількість станів у компанії 1 і у компанії 2 до двох станів, то комбінацій переходів системи буде всього 512, що у 9 разів менше.

- Збільшення або зменшення станів впливає на загальну кількість комбінацій у n^2 разів
- Збільшення або зменшення рішень впливає на загальну кількість комбінацій у n разів
- Додавання або прибирання компанії впливає на загальну кількість комбінацій у $k^2 * n$ разів, де k – кількість станів, а n кількість рішень

Це наглядно показує наскільки сильно може розростатись система при додавання нових компаній, збільшення станів або рішень.

Розділ 3. Програмна реалізація моделі

3.1. Технології, які були використанні

Програма написана на двох мовах програмування, за генерацію і форматування даних і їх подальшого виводу використовується JavaScript MVVM фреймворк Vue(3.2.45) з використанням TypeScript(4.7.4). Також для кращої візуалізації зв'язків компаній використана бібліотека v-network-graph. Після генерації даних і їх нормалізації ми використовуємо мову програмування python і 2 модуля для нього: **scipy** – для пошуку розв'язку задачі лінійного програмування на максимізацію прибутку, та **math** модуль, щоб задати обмеження до нескінченості.

3.2. Методи і змінні програми

Змінні які використовуються:

- **nodes** – двовимірний масив який виступає в ролі матриці зв'язків між компаніями
- **graphNodes**, **graphLayouts**, **graphEdges** – данні, для візуалізації компаній у вигляді графів для бібліотеки v-network-graph
- **node0Reward, node1Reward, node2Reward** – масиви, які виступають в ролі матриць нагород за прийняття рішень
- **rewardsTotal** – масив об'єктів, який зберігає інформацію про стан системи і нагороду за прийнятті рішення
- **probsn0, probsn1, probsn2** – масиви об'єктів, які зберігають інформацію про ймовірність переходу з одного стану в інший для компаній
- **q** – масив об'єктів, які зберігають інформацію про ймовірність переходу з одного стану в інший для системи
- **normalizedLimitsLeft, normalizedLimitsRight –** обмеження для функції лінійного програмування, де використовується знак рівності "більше або дорівнює"

- **normalizedEqLeft, normalizedEqRight** – обмеження для функції лінійного програмування, де використовується знак рівності “дорівнює”
- **normalizedSimplexFunc –** функція лінійного програмування задана в спеціальному форматі, який приймає метод *linprog* з пітонівського модуля *scipy.optimize*
- **result** – відформатовані значення результату обрахунку задачі лінійного програмування, які можна вивезти на екран користувачу

методи які використанні:

- **useGraph** – метод який на вхід приймає параметр **nodes**, який є двомірним масивом зв’язків між компаніями і повертає змінні **graphLayouts, graphNodes, graphEdges**
- **generateRandomProbs** – метод який на вхід приймає параметр **n**, який є числом для скількох станів треба згенерувати ймовірності. Повертає масив чисел з довжиною n, сума яких дорівнює одиниці
- **getProb0, getProb1, getProb2 –** методи які приймають поточний стан компанії, стан компаній від яких залежить, поточне рішення компаній і наступний стан, щоб взяти інформацію з відповідних змінних **probsn0, probsn1, probsn2** про ймовірність переходу в цей стан

3.3. Результат роботи програми

Данна програма моделює систему конкуруючих компаній $N \in \{n1, n2, n3\}$, матриця зв’язку має вигляд $L = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$. Для того, щоб не було великих і важких обрахунків, бо як було вже зазначено раніше, кожен стан збільшує у n^2 разів можливі комбінації, а рішення у k разів, кожна компанія буде мати по два стани: $S1 \in \{s1, s2\}, S2 \in \{s1, s2\}, S3 \in \{s1, s2\}$ і по два рішення $A1 \in$

$\{a1, a2\}\, A2 \in \{a1, a2\}\, A3 \in \{a1, a2\}$. Матриці нагород відповідно буде задана такими значеннями $R1 = (30{,}40), R2 = (10, 50), R3 = (60, 3)$. Ймовірності переходів станів компаній задавати вручну не є доцільно, бо навіть при таких малих значень станів і рішень у компаній, усе одно буде завелика кількість комбінацій, тому ми присвоюємо їм випадково згенеровані значення.

Задача зводиться до того, що нам потрібно вирішити задачу лінійного програмування ось такого виду[15]:

$$\min \sum_{x \in X} \sum_{k=1}^{K} W_{xk}(i) z_{xk}$$

З обмеженнями:

$$\sum_{k=1}^{K} z_{xk} = \sum_{y \in X} \sum_{k=1}^{K} z_{yk} q_{yk}(k), \quad x \in X,$$

$$\sum_{y \in X} \sum_{k=1}^{K} z_{yk} = 1,$$

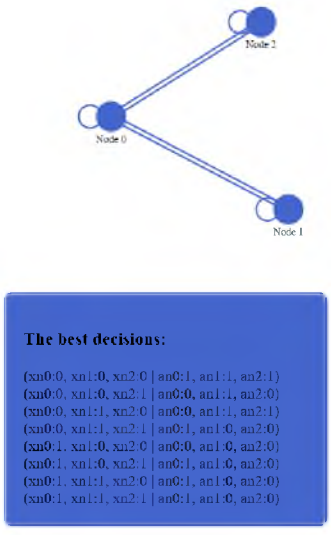
$$z_{xk} \geq 0, \quad x \in X, \quad k = 1, \ldots, K.$$

В ході обчислень симплекс методом цієї задачі ми отримаємо як результат наступну інформацію:

- message – загальна інформація про успіх або помилку обчислень
- success – булеве значення True або False
- status – код статуса виконання функції
- fun – значення лінійного рівняння

- x – масив оптимальних коефіцієнтів, які були обраховані
- nit – кількість ітерацій які знадобились

Нас цікавить тільки x, який ми далі повертаємо у JavaScript для нормалізації і виводу на екран користувача інтуїтивно зрозумілу інформацію.



Висновки

Було розглянуто інформацію про те, що таке марковські моделі, що таке марковська властивість, які є марковські моделі і згодом окремо розглянуто кожну з них, яка в них структура, що вони таке і як їх застосовують у реальному житті, а ми навіть і не замислюємся про це.

Наступне що було розглянуто це загальні факти про конкуренцію, до чого вона призводить, чому вона є важливою і як за допомогою прогнозування можливих рішень конкурентів залишатись не тільки конкурентоспроможним, а й максимізувати свій прибуток, використовуючи марковські моделі поведінки систем з конкуруючими технологіями.

І на останок ми реалізували програму, яка моделює таку систему і обраховує оптимальні стратегії керування. Також було розписано, як в залежності від кількості параметрів системи: кількість компаній, кількість станів в компаній, кількість рішень в компаніях зростають кількість ймовірний подій.

Список використаних джерел

1. Р. К. Чорней Теорія ймовірностей і випадкові процеси Навчальний посібник, Київ 2020
2. Markov models and Markov chains explained in real life: probabilistic workout routine, Dec 31, 2020 by Carolina Bento https://towardsdatascience.com/markov-models-and-markov-chains-explained-in-real-life-probabilistic-workout-routine-65e47b5c9a73
3. Donovan, T. M. and C. Welden. 2002. Spreadsheet exercises in ecology and evolution. Sinauer Associates, Inc. Sunderland, MA, USA.
4. Caswell, H. 2001. Matrix Population models, Second Edition. Sinauer Associates, Inc. Sunderland, MA.
5. Markov and Hidden Markov Model Elaborated with examples, Aug 18, 2020, by Vivekvinushanth Christopher https://towardsdatascience.com/markov-and-hidden-markov-model-3eec42298d75
6. Застосування прихованих марковських моделей до розв'язання задачі розпізнавання акордів, 2021, Андрущак Григорій
7. І. О. Миколюк АНАЛІЗ МЕТОДІВ РОЗПІЗНАВАННЯ МОВЛЕННЯ
8. D.B. Paul .Speech Recognition Using Hidden Markov Models
9. What Is the Markov Decision Process? Definition, Working, and Examples, December 20, 2022, Vijay Kanade https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-markov-decision-process/
10. Real World Applications of Markov Decision Process, Jan 9, 2021, Somnath Banerjee https://towardsdatascience.com/real-world-applications-of-markov-decision-process-mdp-a39685546026
11. POMDP: Introduction to Partially Observable Markov Decision Processes Michael Hahsler and Hossein Kamalzadeh May 20, 2021
12. 9.5.6 Partially Observable Decision Processes, Artificial Intelligence: foundations of computational agents, Cambridge University Press, 2017
13. Solving Imperfect Information Poker Games Using Monte Carlo Search and POMDP Models, 2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)
14. Cybernetics and Systems Analysis, Vol. 47, No. 3, May, 2011 ON MARKOV STOCHASTIC PROCESSES WITH LOCAL INTERACTION FOR SOLVING SOME APPLIED PROBLEMS P. S. Knopova and A. S. Samosonoka

15. Cybernetics and Systems Analysis, VoL 34, No. 3, 1998 SYSTEMS ANALYSIS CONTROL PROBLEMS FOR MARKOV PROCESSES WITH MEMORY P. S. Knopov and R. K. Chornei

Додаток А

(обов'язковий)

Код на JavaScript

```
<template>
  <div>
    <v-network-graph
      class="graph"
      :nodes="graphNodes"
      :edges="graphEdges"
      :layouts="graphLayouts"
    />
    <div class="results">
      <h3><b>The best decisions:</b></h3>
      <p class="p" v-for="res in result" :key="res">
        {{ res }}
      </p>
    </div>
  </div>
</template>

<script setup lang="ts">
import { ref, computed } from "vue";
import { VNetworkGraph } from "v-network-graph";
import useGraph from "@/composables/UseGraph";

const nodes = ref([
```

```
  [1, 1, 1],
  [1, 1, 0],
  [1, 0, 1],
]);

const { graphNodes, graphLayouts, graphEdges } = useGraph(nodes);

const node0Reward = [30, 40];
const node1Reward = [10, 50];
const node2Reward = [60, 3];

const rewardsTotal = computed(() => {
  const rew = [];

  for (let x0 = 0; x0 < 2; x0++) {
    for (let x1 = 0; x1 < 2; x1++) {
      for (let x2 = 0; x2 < 2; x2++) {
        for (let a0 = 0; a0 < 2; a0++) {
          for (let a1 = 0; a1 < 2; a1++) {
            for (let a2 = 0; a2 < 2; a2++) {
              rew.push({
                name: `xn0:${x0} xn1:${x1} xn2:${x2} an0:${a0} an1:${a1} an2:${a2}`,
                sum: node0Reward[a0] + node1Reward[a1] + node2Reward[a2],
              });
            }
          }
```

```
        }
      }
    }
  }

  return rew;
});

function generateRandomProbs(n: number) {
  const probs = [];

  for (let i = 0; i < n; i++) {
    probs.push(Math.floor(Math.random() * 100) + 1); // 1 - 100
  }

  const sum = probs.reduce((prev, curr) => prev + curr, 0);

  return probs.map((item) => Math.round((item * 100) / sum) / 100);
}

const probsn0 = computed(() => {
  const prob = [];
  for (let x0 = 0; x0 < 2; x0++) {
    for (let x1 = 0; x1 < 2; x1++) {
      for (let x2 = 0; x2 < 2; x2++) {
        for (let a = 0; a < 2; a++) {
```

```
        const generatedProb = generateRandomProbs(2);
        for (let nextx0 = 0; nextx0 < 2; nextx0++) {
          prob.push({
            name: `nextx0:${nextx0} x0:${x0} x1:${x1} x2:${x2} a:${a}`,
            prob: generatedProb[nextx0],
          });
        }
      }
    }
  }
  return prob;
});

const probsn1 = computed(() => {
  const prob = [];
  for (let x0 = 0; x0 < 2; x0++) {
    for (let x1 = 0; x1 < 2; x1++) {
      for (let a = 0; a < 2; a++) {
        const generatedProb = generateRandomProbs(2);
        for (let nextx1 = 0; nextx1 < 2; nextx1++) {
          prob.push({
            name: `nextx1:${nextx1} x0:${x0} x1:${x1} a:${a}`,
            prob: generatedProb[nextx1],
          });
        }
```

```
      }
    }
  }
  return prob;
});

const probsn2 = computed(() => {
  const prob = [];
  for (let x0 = 0; x0 < 2; x0++) {
    for (let x2 = 0; x2 < 2; x2++) {
      for (let a = 0; a < 2; a++) {
        const generatedProb = generateRandomProbs(2);
        for (let nextx2 = 0; nextx2 < 2; nextx2++) {
          prob.push({
            name: `nextx2:${nextx2} x0:${x0} x2:${x2} a:${a}`,
            prob: generatedProb[nextx2],
          });
        }
      }
    }
  }
  return prob;
});

function getProb0(x0: number, x1: number, x2: number, a: number, next: number) {
  const prob = probsn0.value.find(
```

```
    (item) => item.name === `nextx0:${next} x0:${x0} x1:${x1} x2:${x2} a:${a}`
  );
  return prob!.prob;
}

function getProb1(x0: number, x1: number, a: number, next: number) {
  const prob = probsn1.value.find(
    (item) => item.name === `nextx1:${next} x0:${x0} x1:${x1} a:${a}`
  );
  return prob!.prob;
}

function getProb2(x0: number, x2: number, a: number, next: number) {
  const prob = probsn2.value.find(
    (item) => item.name === `nextx2:${next} x0:${x0} x2:${x2} a:${a}`
  );
  return prob!.prob;
}

const q = computed(() => {
  const probs = [];

  for (let nextx0 = 0; nextx0 < 2; nextx0++) {
    for (let nextx1 = 0; nextx1 < 2; nextx1++) {
      for (let nextx2 = 0; nextx2 < 2; nextx2++) {
        for (let x0 = 0; x0 < 2; x0++) {
```

```
        for (let x1 = 0; x1 < 2; x1++) {
          for (let x2 = 0; x2 < 2; x2++) {
            for (let a0 = 0; a0 < 2; a0++) {
              for (let a1 = 0; a1 < 2; a1++) {
                for (let a2 = 0; a2 < 2; a2++) {
                  probs.push({
                    name: `nextx0:${nextx0} nextx1:${nextx1} nextx2:${nextx2} xn0:${x0}
xn1:${x1} xn2:${x2} an0:${a0} an1:${a1} an2:${a2}`,
                    prob:
                      getProb0(x0, x1, x2, a0, nextx0) *
                      getProb1(x0, x1, a1, nextx1) *
                      getProb2(x0, x2, a2, nextx2),
                  });
                }
              }
            }
          }
        }
      }
    }
  }

  return probs;
});

const normalizedLimitsLeft = computed(() => {
```

```
  const arr: number[][] = [];

  for (let i = 0; i < q.value.length / 8; i++) {
    let buffer: number[] = Array(q.value.length).fill(0, 0, q.value.length);
    for (let j = 0; j < 8; j++) {
      buffer[i * 8 + j] = +q.value[i * 8 + j].prob.toFixed(3) * -1;
    }
    arr.push(buffer);
  }

  return arr;
});

const normalizedLimitsRight = computed(() => {
  const arr: number[] = [];

  arr.push(...Array(q.value.length / 8).fill(0, 0, q.value.length / 8));

  return arr;
});

const normalizedEqLeft = computed(() => {
  const arr: number[][] = [];

  for (let i = 0; i < q.value.length / 8; i++) {
    let buffer: number[] = Array(q.value.length).fill(0, 0, q.value.length);
```

```
    for (let j = 0; j < 8; j++) {
      buffer[i * 8 + j] = 1;
    }
    arr.push(buffer);
  }

  return arr;
});

const normalizedEqRight = computed(() => {
  const arr: number[] = [];

  arr.push(...Array(q.value.length / 8).fill(1, 0, q.value.length / 8));

  return arr;
});

const normalizedSimplexFunc = computed(() => {
  return rewardsTotal.value
    .reduce((prev, curr, idx) => {
      let xParser = "";
      for (let i = 0; i < 8; i++) {
        xParser += `${(q.value[idx * 8 + i].prob * curr.sum).toFixed(3)}`;
        if (i < 7) {
          xParser += ", ";
        }
```

```
      }

      if (prev) {
        return `${prev}, ${xParser}`;
      }
      return `${xParser}`;
    }, "")
    .trim()
    .split(", ")
    .map((numb) => +numb);
});

const simplexResult = opt.x з результатів обрахунків python

const result = computed(() => {
  const buffer = q.value
    .map((item, idx) => ({
      name: item.name,
      isBestDecision: !!simplexResult[idx],
      prob: item.prob,
    }))
    .filter((item) => item.isBestDecision)
    .map((item) => {
      const an0 = +item.name.split("an0:")[1].split(" ")[0];
      const an1 = +item.name.split("an1:")[1].split(" ")[0];
      const an2 = +item.name.split("an2:")[1].split(" ")[0];
```

```
    const value =
      item.prob * (node0Reward[an0] + node1Reward[an1] + node2Reward[an2]);
    return {
      name: item.name.split(" ").slice(3).join(" "),
      value,
    };
  })
  .sort((a, b) => {
    if (a.name < b.name) {
      return -1;
    }
    if (a.name > b.name) {
      return 1;
    }
    return 0;
  });

let matrix = new Array(8).fill(null).map(() => new Array(8).fill(null));

for (let i = 0; i < 8; i++) {
  for (let j = 0; j < 8; j++) {
    matrix[i][j] = buffer[i * 8 + j];
  }
}
```

```
  matrix = matrix.map((subarr) => {
    let maxIndex = 0;

    for (let i = 1; i < subarr.length; i++) {
      if (subarr[i].value > subarr[maxIndex].value) {
        maxIndex = i;
      }
    }

    return subarr[maxIndex];
  });

  return (
    matrix as unknown as {
      name: string;
      value: number;
    }[]
  ).map((item) => {
    const spilt = item.name.split(" ");

    return `(${spilt[0]}, ${spilt[1]}, ${spilt[2]} | ${spilt[3]}, ${spilt[4]}, ${spilt[5]})`;
  });
});
</script>

<style scoped>
```

```
.graph {
  width: 99vw;
  height: 50vh;
}

.results {
  width: 300px;
  margin-left: 20px;
  padding: 20px;
  border-radius: 8px;
  background-color: #4466cc;
}
</style>
```

Код на python

```
from scipy.optimize import linprog
import math

obj = normalizedSimplexFunc з JavaScript

lhs_ineq = normalizedLimitsLeft з JavaScript

rhs_ineq = normalizedLimitsRight з JavaScript

lhs_eq = normalizedEqLeft з JavaScript

rhs_eq = normalizedEqRight з JavaScript
```

```
bnd = [(0, math.inf), …] # 512 елементів (0, math.inf)


opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq, A_eq=lhs_eq, b_eq=rhs_eq,
bounds=bnd, method="revised simplex")


opt
```