Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«КЕРУВАННЯ В МЕРЕЖАХ ҐОРДОНА-НЬЮВЕЛА»**

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

Демченко Роман Олегович

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

Рецензент_____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2022

Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
проф., д.ф-м.н., Б. В. Олійник
_____
(підпис)
„____” _____2022
р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу

студенту 4-го курсу, факультету інформатики__
Демченку Роману Олеговичу_____
Розробити Модель керування та симуляції в мережах Ґордона-Ньювела

Зміст ТЧ до кваліфікаційної роботи:
Зміст
Анотація
Вступ
1 Огляд теоретичної частини
2 Постановка задачі
3 Постановка алгоритму задачі
4 Розробка алгоритму програми
5 Результати тестувань
Додатки
Висновки
Список літератури

Дата видачі „___” _____ 2022 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Додаток Г
(обов'язковий)
Зразок календарного плану виконання курсової роботи

Тема:

_____

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	02.11.2021	
2.	Огляд технічної літератури за темою роботи.	15.11.2021	
7.	Написання пояснювальної роботи.	20.04.2022	
8.	Створення слайдів для доповіді та написання доповіді.	22.04.2022	
8.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист кваліфікаційної роботи.	15.06.2022	
10.	Корегування роботи за результатами попереднього захисту.	18.06.2022	
11.	Остаточне оформлення пояснювальної роботи та слайдів.	29.06.2022	
12.	Захист кваліфікаційної роботи	04.07.2022	

Студент _____

Керівник _____

"_____" _____

Зміст

Анотація

В роботі досліджується поведінка в мережах Ґордона - Ньювела, розглядається теорія черг, теорема Джексона. В результаті роботи запропонована модель керування мережами Ґордона - Ньювела, також реалізована симуляція мережі в реальному часі.

Ключові слова: оптимальні стратегії, теорія черг, мережі Ґордона - Ньювела.

2 Вступ

2.1 Актуальність

Багато систем, що можуть бути описані як мережі масового обслуговування, можна оптимізувати за допомогою теореми Ґордона - Ньювела, а саме, це: підвищення їх продуктивності, розрахунок цінних характеристик, і загалом знаходження слабких місць у системі. Прикладів таких систем існує безліч: супермаркет, інтернет-магазин, фабрика, шиномонтаж і т. д – все це системи, в яких необхідно формування черги. Будь-яка черга сама по собі - це модель, де прогнозується довжина (в її метриці), та час очікування в момент часу.

Теорія масового обслуговування як раз і створює моделі, що обчислюють, наприклад, час очікування, та підвищують ефективність системи обслуговування. У 1967 році Гордон Вільям Джон та Гордон Френк Ньюел опублікували статтю, в якій досліджували закриті мережами масового обслуговування експоненційних серверів, що є окремим випадком мереж масового обслуговування. [1]

2.2 Об’єкт дослідження

Мережі Ґордона-Ньювела.

2.3 Предмет дослідження

Моделювання, та керування мережею Ґордона-Ньювела.

2.4 Мета, завдання дослідження

Моделювання керованих випадкових процесів з неперервним часом на прикладі узагальненої мережі Ґордона-Ньювела.

2.5 Методи дослідження

Теорія масового обслуговування, теорія ймовірностей, теорія керування, теорія випадкових процесів.

1.6 Опис роботи

Робота складається з розділів **2**, **3**; В розділі **2** описуються теоретичні аспекти моделювання та керування мережею Ґордона-Ньювела, її основні властивості. В розділах **2.1**, **2.2** розглядається необхідний фундамент для глибшого розуміння теми. Розділ **3** є практичною частиною роботи. В ньому аналізується побудована модель, та керування нею.

3 Мережі Ґордона-Ньювела

3.1 Загальні відомості з теорії масового обслуговування (теорії черг)

3.1.1 Існує три категорії мереж з теорії черг:

1) Відкриті мережі – це найпоширеніші мережі, в них нові користувачі можуть увійти в мережу, та вийти з неї. Користувачі цієї мережі можуть переходити з одного вузла до іншого.

2) Закриті мережі – мають завжди константну кількість користувачів, кожен користувач час від часу змінює свій вузол.

3) Змішані – це мережа, що є відкритою для певного класу користувачів, та закритою для інших.

3.1.2 Існують наступні типи систем масового обслуговування:

Теорія масового обслуговування використовує нотацію Кендалла для класифікації черги, та її характеристик. Загалом черга описується за наступним позначенням (2), де:

A – процес “прибуття”, позначає час між прибуттями у чергу.

S – математичний розподіл загального часу обслуговування.

c – кількість серверів (вузлів).

K – кількість черг (пропущена, якщо вона необмежена).

N – кількість можливих клієнтів (пропущена, якщо необмежена).

D – це порядок черги, яка передбачається за алгоритмом першим прийшов – першим вийшов.

3.1.3 Закон Літтла (Little’s Law)

Закон Літтла пов’язує потужність системи масового обслуговування, середній час, проведений в системі, і середню швидкість надходження в систему, не знаючи жодних інших особливостей черги. Формула досить проста і записується так:

$$L = \lambda W$$

– Звідки:

L – середня кількість клієнтів у системі.

λ – середня швидкість надходження клієнтів у систему.

W – середня кількість часу, який клієнт проводить в системі.

Закон Літтла критично важливий для бізнес-додатків, у яких його можна написати простою формулою:

$$\text{пропускна спроможність} = \frac{\text{кількість задач}}{\text{час перебування в системі}}$$

Наприклад, якщо ви чекаєте в черзі в Starbucks, закон Літтла може оцінити, скільки часу знадобиться, щоб отримати вашу каву. [4]

Припустимо, що в черзі 15 осіб, один продавець (вузол) і 2 особи обслуговуються за одну хвилину. Щоб оцінити очікування, ви повинні використати закон Літтла у вигляді:

$$\frac{\text{15 осіб в черзі}}{\text{2 особи в хв}} = 7.5 \text{ хвилин очікування}$$

– Закон Літтла є одним із фундаментальних законів теорії масового обслуговування.

3.2 Теорема Джексона

Оскільки теорема Ґордона - Ньювела є розширенням теореми Джексона, то почнемо ми саме з неї. Мережа Джексона – це клас мережі масового обслуговування, де рівноважний розподіл особливо легко обчислити, оскільки мережа має рішення у формі добутку.

Мережа з m взаємопов'язаних черг називається мережею Джексона (або ж Джексонівською мережею), якщо вона відповідає наступним умовам: [5]

1) Якщо мережа закрита, будь-які зовнішні надходження до вузла i утворюють процес Пуассона.

2) Увесь час обслуговування розподілене експоненційно, та порядок обслуговування в усіх чергах передбачається за алгоритмом: першим прийшов – першим вийшов.

3) Клієнт, що виходить з черги i, або переміщається до деякої нової черги j з ймовірністю Pij, або виходить з системи з ймовірністю $\left(1 - \sum_{j=1}^{m} Pij \right)$ – яка для відкритої мережі $\neq 0$, в деякій підмножини черг.

4) “Утилізація” всіх черг < 1.

– Під утилізацією (Rental Utilization) розуміється, що це є певною мірою фактичної кількості, що протиставляється потенційній кількості клієнтів у вузлі, що були в черзі попереднього разу. Тобто, можна сказати, що утилізація – це зріст клієнтів з кожним кроком. [7]

У відкритій мережі Джексона з m черг M/M/1 (1), де “утилізація” $\rho i < 1$ у кожній черзі, існує розподіл ймовірності стану рівноваги, а для стану $(k_1, k_2, \ldots, k_m)$ визначається добутком рівноважних розподілів окремих черг:

$$\pi(k_1, k_2, \ldots, k_m) = \prod_{i=1}^{m} \pi_i(k_i) = \prod_{i=1}^{m} \left[p_i^{k_i} (1 - p_i) \right]$$

– Наступний результат також стосується моделей M/M/c (2) із серверами c_i вузлі, з наступною вимогою “утилізації”: $p_i < c_i$;

(1)* – У нотації Кендалла, модель описує систему, де прибуття користувачів утворюють одну чергу що керується процесом Пуассона. Час обслуговування завдань розподіляється експоненціально. А c – позначає сервери (вузли).

(2)* – Черга M/M/1 представляє довжину черги в системі с одним сервером (вузлом). Прибуття користувачів також визначається розподілом Пуассона, а час обслуговування має експоненційний розподіл.

3.3 Властивості мереж Ґордона – Ньювела

Типова мережа Ґогдона – Ньювела (або ж закрита мережа Джексона) має наступні властивості:

1) Є відкритою, сильнозв'язною мережею, з J вузлами (чергами), та загальною кількістю користувачів K, що циркулюють по ній.

2) Увесь час обслуговування розподілене експоненційно, та порядок обслуговування в усіх чергах передбачається за алгоритмом: першим прийшов – першим вийшов.

3) Клієнт, що завершує чергу i, переходить у чергу j з ймовірністю P_{ij} такою, що: $\sum_{j=1}^{m} P_{ij} = 1$;

4) "Утилізація" всіх черг < 1.

Поточний стан мережі описуємо наступним простором станів:

$$S(K, J) = (n_1, n_2, \ldots, n_J)$$

– Де n_i це кількість користувачів в черзі вузла i.

Тоді, відповідно, загальна кількість користувачів (K) у момент t буде:

$$\sum_{i=1}^{J} n_i = K$$

Тоді, розподіл ймовірності стану рівноваги існує і визначається так: [6]

$$\pi(I, J)(k_1, k_2, \ldots, k_m) = \frac{1}{G} \prod_{i=1}^{J} \left\{ \prod_{i=1}^{J} \frac{\eta_j}{\mu_j} \right\}$$

– Звідси час обслуговування в черзі i експоненційно розподілено з параметром μ_i. Також передбачається, що середній час обслуговування вузла не залежить від кількості присутніх клієнтів у ньому. З формули ще маємо нормуючу константу G, вона визначається наступним чином: [6]

$$G = \sum_{n=0}^{J} \prod_{i=1}^{n} \frac{\lambda(i-1)}{\mu(i)}$$

З теореми маємо останнє: коефіцієнт відвідування e_i, що розраховується шляхом розв'язування одночасних рівнянь: [6]

$$e_i = \sum_{i=1}^{m} e_i p_{ij} - \text{для } i \in [1;\ m];$$

3.4 Доповнена мережа Ґордона – Ньювела

В доповнених мережах Ґордона - Ньювела, крім матриці Маркова, додається вектор спільний вектор довжини черги доповненої мережі у момент часу t.$\eta_j^t = (\nu_j^t;\ \vartheta_j^t;\ \varrho_j^t)$, звідси: ν_j^t – кількість клієнтів у вузлі j в момент часу t. ϑ_j^t – рішення, що приймає користувач в момент наступного свого переходу, чи покидати йому вузол. ϱ_j^t – напрямок, та саме вузол, в який користувач буде проходити після свого обслуговування. [2]

Мережа розвивається з часом, та залишається в стані $\eta_j^t = (n_1,\ t_1,\ r_1;\ \ldots;\ n_i,\ t_i,\ r_i;\ \ldots;\ n_j,\ t_j,\ r_j;\ \ldots;\ n_J,\ t_J,\ r_J) > 0$ протягом часу, розподіленого експоненційно з параметром: $\lambda(n_1 \ldots,\ n_i,\ \ldots,\ n_j,\ \ldots, n_J) > 0$. Протягом цього часу λ у вузлах користувачами приймаються рішення, що визначає $\alpha_k^t \in \{0,\ 1\},\ k$ – вузол. [2]

Тобто, кожного разу, для кожного клієнта що був обслужений в деякому вузлі, згідно з ймовірністю $\alpha_i^t = \{0,\ 1\}$ вирішується, чи було його обслуговування успішним. Коли закінчується момент рішення α_i^t (що у реаліях є одномоментним), вузли, пов`язані з вузлом в якому приймалось рішення – а це сам вузол i, та перехідний вузол з $i \to j$ оновлюються: ν_i^{t+1} та ν_j^{t+1}. Останньою операцією буде слідує оновлення $\varrho_j^{t+1} = k$, за ймовірністю $r(j,\ k)$ – з матриці переходів. Проте слід зауважити, що вузол призначення k з напрямку $\varrho_j^{t+1} = k$, з часом не змінюється, та є завжди детермінованим матрицею переходів.

Рішення обслуговування α_i^t та ϑ_j^t – рішення, що приймає користувач в момент переходу є незалежними від історії мережі, та від інших рішень в мережі.

Це дає мережі асинхронність та недетермінованість. [2]

3.5 Керування в мережі Ґордона – Ньювела

Керування в мережах Ґордона-Ньювела відбувається за обраним критерієм керування. Критерієм керування можуть бути середні витрати в одиницю часу – це, наприклад, загальна кількість енергії, спожита вузлами. Або ж, рівновага станів системи. Обраним в роботі критерієм є швидкість обслуговування всієї бази клієнтів.

Керування – це набір дій, в залежності від яких маємо різні результати введеного критерію керування. Для оптимального керування обираються такі дії, що призводять до оптимального значення критерію керування. Згідно з обраним критерієм, оптимальним керуванням вважається таке, що буде набувати своїх мінімальних значень.

Обраний критерій пов’язується з математичним сподіванням M по випадковій величині, що є сукупністю параметрів мережі та обраної матриці переходів. В загальному випадку мережа є керованою, якщо в ній існує декілька матриць Маркова (переходів), та математичне сподівання кожної з матриць (моделей) відображає критерій керування. Керування проходить таким чином: по кожній з матриць рахується математичне сподівання, та згідно з критерієм обирається оптимальна матриця переходів.

Для керування в цій роботі обчислюються різні стратегії, та після цього повертається оптимальне керування. Для цього, обчислюючи одну стратегію, обчислюємо мат. сподівання по керованим характеристикам та константним характеристикам системи. З висновку попереднього розділу слідує, що мережа є асинхронною. Аби розрахувати цей не дискретний процес, я розбив задачу на покрокові обчислення очікуваної кількості спроб перейти, та відповідно часу переходу кожного клієнта з вузла $i \rightarrow j$, згідно з вектором direction (див. далі) та пов’язаних з цим переходом змін параметрів мережі. Оскільки мережа працює в асинхронному режимі, маємо розраховувати очікуваний час в кожному вузлі паралельно, якщо вузол є зайнятим в момент. Для паралельних обчислень в роботі знаходиться такий наступний перехід, що буде мінімальним. Очікуваний час цього переходу віднімається від часу кожного з поточних процесів переходу – так, ніби він був виконаний паралельно з ними. Після цього відбувається перехід клієнта в інший вузол (або вихід з мережі).

Загальний час обслуговування моделі P_n якраз складається з суми кожного з цих “маленьких” послідовних переходів.

Один перехід користувача з вузла $i \ -> \ j$ оцінюємо так:

$$E(\mu;\ I;\ p(i,\ j)) = \ F(p(i,\ j)) \ \ * \ \mu(i) \ * e^{I}$$

$$F(p(i,\ j)) \ = \sum_{n=1}^{k} p(i,\ j)^{n} = \ 1 \ => \ k$$

– тобто така функція F повертає k, де k – кількість спроб, не цілочисельна.

де: $\mu(i)$ час обслуговування у вузлі i;

$p(i,\ j)$ – елемент матриці маркова, з $i \ -> \ j$;

I– кількість користувачів в мережі.

В мережі оцінюються всі такі можливі переходи паралельно. Це обчислюється як: min – мінімум по всім таким переходам, після чого від очікуваного часу кожного переходу H(i - вузол) віднімається min. Таким чином, в моменти часу таких “стрибків”, один з переходів H(i) завжди буде дорівнювати 0. Тоді, за цей крок обчислюємо E для {H(i) = 0}, та знову віднімаємо min по всім H(i). Таким чином, обчислення тривають ніби паралельно, чергуючи обчислення E для {H(i) = 0} і {H(0..J) - min}. Це триває поки {(I > 0) ^ (N > 0)}, де N – кількість клієнтів в мережі.

Також інші важливі компоненти системи, без яких моделювання, і отже керування є неможливим:

Незвідна матриця Маркова є імовірнісною репрезентацією переходу клієнтів з вузла i, у вузол j: r(i, j) - елемент.

μ_j – задає швидкість обслуговування клієнта у вузлі j. ϑ_j= {0, 1} – рішення, що приймає користувач після закінчення свого обслуговування у вузлі j.

Рішення 1 – перехід у інший вузол (або вихід з мережі), 0 – повторне обслуговування в мережі, та пересування у кінець черги. ϱ_j – напрямок (вузол), який обирає клієнт, по закінченню свого обслуговування у вузлі j.

Зазвичай перехід (i –> j) задається для всіх клієнтів мережі поголовно. Q – матриця інтенсивностей переходів. Представляє собою матрицю n*n, де n – кількість вузлів у графі станів марківської системи.

λ – експоненційний розподіл, за яким задається обслуговування клієнтів, та їх прибуття в систему. Розподіл задає час обслуговування в вузлах (серверах), та швидкість прибуття в мережу що залежать від кількості користувачів в мережі. Тобто, чим більше користувачів – тим більше навантаження на мережу, та кожний її вузол.

π – стаціонарний граничний розподіл, з нормуючою константою G.

Для існування розподілу π в деякій мережі, необхідно, аби її спільний вектор довжини $\eta_{i...j}$ не мав значень меньше нуля.

4 Моделювання мережі Ґордона – Ньювела

Побудована модель є симуляцією мережі Ґордона-Ньювела в реальному часі. Також передбачається керування мережею за критерієм середнього очікуваного часу обслуговування всіх клієнтів в мережі. Аспекти керування окреслені в розділі **2.4**;

Використана мова Python, усе важливе, окрім алгоритму, окреслене коментарями в коді. Додаток міститься нище.

Приклад моделі:

```
За критерієм керування маємо такі очікувані стратегії:  [111.24666360664581, 14.741494038684305, 29.251032134576345]
З них обираємо найкращу (найменшу - найшвидшу, за введеним критерієм):  14.741494038684305
Тоді за даних умов:
Швидкість обслуговування в вузлі j:  [0.02 0.05 0.05]
Загальна кількість користувачів:  4
Експоненційний розподіл, задає час обслуговування в залежності від кількості користувачів в системі у момент часу:  [

Обираємо стратегію № 2
Та відповідну їй матрицю Маркова:
 [[0.  0.5 0.5]
 [0.2 0.  0.8]
 [0.4 0.4 0.2]]

Симуляція мережі заданою за цією стратегією:
3 -> [1 0 0]; pi = [0 0 0]

2 -> [2 0 0]; pi = [0 0 0]

2 -> [1 1 0];  decision = [1, '-', '-'];  direction = [1, '-', '-'];  pi = [0 0 0];

1 -> [2 1 0]; pi = [0 0 0]

1 -> [2 1 0];  decision = [0, '-', '-'];  direction = ['-', '-', '-'];  pi = [0 0 0];

0 -> [3 1 0]; pi = [0 0 0]

0 -> [3 0 1];  decision = ['-', 1, '-'];  direction = ['-', 2, '-'];  pi = [0 0 0];

0 -> [2 1 1];  decision = [1, '-', '-'];  direction = [1, '-', '-'];  pi = [        0     18823 752941176];

0 -> [1 2 1];  decision = [1, '-', '-'];  direction = [1, '-', '-'];  pi = [        0     18823 752941176];
```

– За критерієм модель оцінює стратегії, та обирає найкращу (№2);

Процес симуляції показовий:

Наприклад маємо наступний стан у момент часу t: 2 → [2 1 0]; desition [0, ‘-’, ‘-’]; direction;

– Перша цифра це черга клієнтів, що очікують своєї черги. Вектор типу [1, 0, 0] – це спільний вектор довжини черги. Вектор decision[i] = j, де i – це вузол, який відвідав користувач, та j – це рішення, яке він приймає.

‘-’ – рішення з вузла не надходило, 0 – повторне відвідування поточного вузла, 1 – продовження руху по мережі згідно з заданим вектором переходів direct (див. у додатку нище).

direction – це направлення поточного клієнта, що тільки що був обслугований. direction[i] = j – де i - це вузол з якого виходить клієнт, та j – в котрий він направляється. pi – стаціонарний і граничний розподіл.

В роботі задані наступні керовані компоненти керування: матриця Маркова, швидкість обслуговування у вузлах, кількість користувачів на стратегію. Інші змінні є константними. Керування в роботі відбувається таким чином: кожна з стратегій оцінюється відповідно за критерієм керування, в цій роботі – це швидкість обслуговування всієї бази клієнтів. Кожна стратегія оцінюється за математичним очікуванням часу обслуговування всіх клієнтів, тобто, до стану системи {I = 0, N = 0}, де I – кількість не обслужених клієнтів поза системою, а N – кількість клієнтів у системі. Докладніше про обчислення мат. очікування читайте в попередньому розділі **2.4**

Для демонстрації процесу керування було обрано наступні три стратегії:

```
P1 = np.array(
    [[0, 0.3, 0.7],
     [0.7, 0, 0.3],
     [0.4, 0.4, 0.2]],
) P2 = np.array(
    [[0, 0.5, 0.5],
     [0.2, 0, 0.8],
     [0.4, 0.4, 0.2]],
) P3 = np.array(
    [[0, 0.3, 0.7],
     [0.3, 0, 0.7],
     [0.4, 0.4, 0.2]],
)

mu_strategies = np.array(
    [[0.03, 0.08, 0.04], # <-- стратегія 1
    [0.02, 0.05, 0.05],  # <-- стратегія 2
    [0.05, 0.03, 0.06]], # <-- стратегія 3
)

I_strategies = np.array([5, 4, 5])
```

– Де P1, P2, P3 - матриці Маркова; mu_strategies[i] – це три стратегії швидкості обслуговування у відповідному вузлі [i][j], а I_strategies[i] – кількість користувачів в мережі на на стратегію і. Тобто, кожна стратегія являє собою набір параметрів P[i], mu[i], I[i]; Керування ж відбувається за оцінки кожної стратегії, та вибором кращої, за критерієм керування.

Також решта константних параметрів:

Експоненціально розподілений вектор λ = (0.01, 0.02718, 0.07389, 0.20085, 0.5459);

Вектор переходів: **direct = [1, 2];** З нього маємо наступні переходи: 0 → 1; 1 → 2; 2 → (вихід з мережі за ймовірністю). Деталі окреслені в програмі.

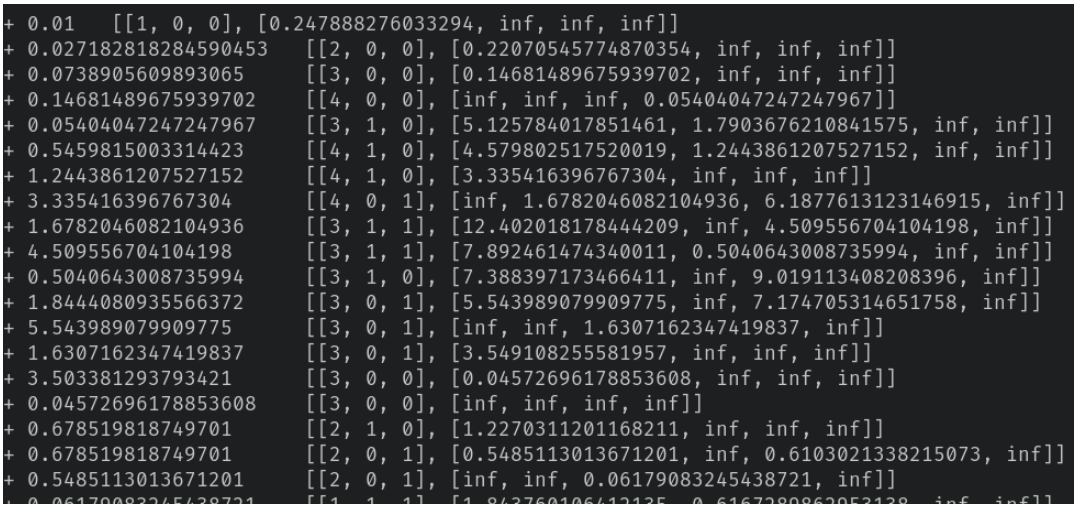
На прикладі цих трьох стратегій побудовані їхні математичні очікування [P1, P2, P3] відповідно:

Результат керування: [111.24666360664581, 14.741494038684305, 29.251032134576345]

– де чисельна характеристика це очікуваний час обслуговування моделі P[i].

Звідки бачимо, що модель P2 є вдвічі продуктивнішою за P3, та у 8 разів за P1, хоча, мат. сподівання повертає лише приблизний час.

Наступне зображення є процесом знаходження мат. очікування моделі. Деталі див. у попередньому розділі.



Звідки ліве значення – покрокові обчислення, що фактично є мінімальним очікуваним часом до наступного переходу користувача в інший вузол.

Вектор посередині – спільний вектор довжини черги. А справа – очікуваний час обслуговування користувача у вузлі [i].

Перевіримо достовірність одержаного мат. очікування з P2:
Було послідовно змодельовано 10 моделей P2, та обчислено час обслуговування кожної, згідно з обраним критерієм керування.

Стратегія 1:

1) Time elapsed: 66.93956351280212
2) Time elapsed: 50.369558811187744
3) Time elapsed: 117.5552990436554
4) Time elapsed: 68.17448091506958
5) Time elapsed: 181.4462366104126

Стратегія 2:
1) Time elapsed: 10.426860570907593
2) Time elapsed: 6.0918495655059814
3) Time elapsed: 5.8376147747039795
4) Time elapsed: 8.568345069885254
5) Time elapsed: 7.416201114654541

Стратегія 3:
1) Time elapsed: 40.8118839263916
2) Time elapsed: 33.05819749832153
3) Time elapsed: 33.14994978904724
4) Time elapsed: 64.99657654762268
5) Time elapsed: 17.38479232788086

Знайдемо середнє арифметичне з двох стратегій:
Стратегія 1: **96.8970277786**
Стратегія 2: **7.66817421913**
Стратегія 3: **37.8802800179**

– Відповідно, за такої різниці, можна стверджувати, що знайдене мат. очікування є в міру точним, та модель здійснює керування за оголошеним критерієм.

5 Додатки

```
import math as math
import numpy as np
import time as time
import random as rnd
import sys
import asyncio
import warnings
warnings.filterwarnings("ignore")

# Ініціалізуємо спільний вектор довжини черги у момент неперервного часу
servers = np.array([0, 0, 0]) # <- мережа з трьома вузлами

I = 5 # кількість клієнтів поза системою, в очікуванні
N = sum(servers) # N позначатимемо за к-ть клієнтів у системі
J = len(servers) # кількість вузлів

# Задаємо незвідну матрицю Маркова
P = np.array(
    [[0, 0.3, 0.7],
     [0.3, 0, 0.7],
     [0.4, 0.4, 0.2]],
)

# Задаємо швидкість, з якою клієнт обслуговується у вузлі j
mu = np.array([0.03, 0.08, 0.04])

# Рішення, що приймає клієнт щодо того, чи покидати йому вузол
epsilon = ['-', '-', '-']

# Напрямок, куди прямує клієнт з вузла i (direction[i] = j) -- де i - це вузол відправлення, а j -- прибуття
direction = ['-', '-', '-']
```

```
# За допомогою змінної direct можна задати маршрутизацію, за якою будуть слідувати користувачі змодельованої мережі
direct = np.array([1, 2]) # direct[i] = j -- де i позначає вузол відправки, а j -- прибуття; в цьому прикладі користувачі виходять з мережі через вузол 3, тому довжина вектора direct = 2

# energy = [1, 2, 3] # енергія, яку споживає вузол при роботі. звідси: energy[i] = j, де - i номерація вузла, j - енергія яку він споживає в секунду реального часу

# alpha distributed exponentially
alpha = np.zeros(I + sum(servers))
for i in range(I + sum(servers)):
    alpha[i] = 0.01 * math.exp(i)

start = time.time() # обрахуємо час, за який пройде одна ітерація симуляції

# Нормуюча константа
G = 0
for n_ in range(J):
    res = 1
    for i in range(1, n_):
        res *= alpha[i - 1] / mu[i]

    G += res


def calculate_pi_distribution():

    if I == 0 and sum(servers) == 0:
        print(f'{I} -> {servers};  decision = {epsilon};  direction = {direction}')
        print("\nTime elapsed: ", time.time() - start)
        sys.exit()
```

```
    # Ініціалізуємо стаціонарний граничний розподіл
    pi = np.array([0, 0, 0])
    n_, j = 0, 0
    # Задаємо стаціонарний граничний розподіл
    for n_ in range(J):
        res = 1
        for j in range(J):
            res2 = 1
            for k in range(n_):
                res2 *= servers[j] / mu[j]
            res *= res2

        pi[n_] = (G ** (-1)) * res

    return pi


# Симуляція мережі
async def arrival():
    global I
    while I > 0:
        await asyncio.sleep(alpha[sum(servers)])
        servers[0] += 1
        I -= 1
        print(f'{I} -> {servers}; pi =
{calculate_pi_distribution()} \n')

    return 0


async def server1():
    global I
    while (True):

        if servers[0] > 0:
            await asyncio.sleep(mu[0] *
math.exp(sum(servers)))
            if rnd.uniform(0, 1) <= P[0][1] and
servers[0] != 0:
```

```
                servers[0] -= 1
                servers[direct[0]] += 1
                epsilon[0] = 1
                direction[0] = direct[0]

            elif servers[0] != 0:
                epsilon[0] = 0

            print(f'{I} -> {servers};  decision =
{epsilon};  direction = {direction};  pi =
{calculate_pi_distribution()}; \n')
            epsilon[0] = '-'
            direction[0] = '-'
        await asyncio.sleep(0.00001)

    return 0

async def server2():
    global I
    while (True):

        if servers[1] > 0:
            await asyncio.sleep(mu[1] *
math.exp(sum(servers)))
            if rnd.uniform(0, 1) <= P[1][2] and
servers[1] != 0:
                servers[1] -= 1
                servers[direct[1]] += 1
                epsilon[1] = 1
                direction[1] = direct[1]

            elif servers[1] != 0:
                epsilon[1] = 0

            print(f'{I} -> {servers};  decision =
{epsilon};  direction = {direction};  pi =
{calculate_pi_distribution()}; \n')
            epsilon[1] = '-'
            direction[1] = '-'
```

```
        await asyncio.sleep(0.00001)

    return 0

async def server3():
    global I
    while (True):

        if servers[2] > 0:
            await asyncio.sleep(mu[2] *
math.exp(sum(servers)))
            if rnd.uniform(0, 1) >= P[2][2] and
servers[2] != 0:
                servers[2] -= 1
                epsilon[2] = 1

            elif servers[2] != 0:
                epsilon[2] = 0

            print(f'{I} -> {servers};  decision =
{epsilon};  direction = {direction};  pi =
{calculate_pi_distribution()}; \n')
            epsilon[2] = '-'
            # тут, на відміну від інших вузлів, рахуємо
ймовірність залишитись у сервері 3 (тобто ймов. переходу
3 -> 3)
        await asyncio.sleep(0.00001)

    return 0

# критерій керування: середній час обслуговування всіх
клієнтів

def probability(p_i, k):
    if k == 1:
        return p_i
    else:
        return p_i**k + probability(p_i, k - 1)
```

```
def expect(p_i): # кількість спроб користувача в середньому аби пройти заданий вузол
    result = 0.0
    k = 0.0
    while(result < 1):
        if (result > 0 and result + (k + 1) * probability(p_i, k + 1) > 1):
            leftover = (k + 1) * probability(p_i, k + 1)
            result += leftover
            waste = result - 1
            k += 1 - (waste / leftover)
            return k

        k += 1
        result += k * probability(p_i, k)

def evaluate(P, mu, I):
    system = [[0, 0, 0], [float('inf'), float('inf'), float('inf'), float('inf')]]
    I_cp = I
    skip = True
    result = 0
    while (sum(system[0]) != 0 or I_cp != 0):

        if (I_cp != 0) and ( system[1][len(system[1]) - 1] == float('inf') ):
            system[0][0] += 1
            system[1][len(system[1]) - 1] = alpha[sum(system[0]) - 1]
            I_cp -= 1

        for j in range(len(system[0])):
            if system[0][j] > 0 and system[1][j] == float('inf'):
                if j != len(system[0]) - 1:
                    system[1][j] = expect(P[j][direct[j]]) * mu[j] * math.exp(sum(system[0]))
```

```
                else:
                    system[1][j] = expect(1 - P[j][j])
* mu[j] * math.exp(sum(system[0]))

                if skip:
                    skip = False

                else:
                    system[0][j] -= 1
                    if j != len(system[0]) - 1:
                        system[0][j + 1] += 1
                        system[1][j] =
expect(P[j][direct[j]]) * mu[j] *
math.exp(sum(system[0]))
                        if j != len(system[0]) - 1:
                            skip = True

        if (I_cp != 0) and ( system[1][len(system[1]) -
1] == float('inf') ):
            system[0][0] += 1
            system[1][len(system[1]) - 1] =
alpha[sum(system[0]) - 1]
            I_cp -= 1

        minimum = min( [num for num in system[1]] )
        next_to_move = system[1].index(minimum)
        system[1][next_to_move] = float('inf')
        system[1] = [num - minimum for num in
system[1]]
        result += minimum

    return result


P1 = np.array(
    [[0, 0.3, 0.7],
     [0.7, 0, 0.3],
     [0.4, 0.4, 0.2]],
)
```

```
P2 = np.array(
    [[0, 0.5, 0.5],
     [0.2, 0, 0.8],
     [0.4, 0.4, 0.2]],
)
P3 = np.array(
    [[0, 0.3, 0.7],
     [0.3, 0, 0.7],
     [0.4, 0.4, 0.2]],
)

mu_strategies = np.array(
    [[0.03, 0.08, 0.04], # <-- стратегія 1
    [0.02, 0.05, 0.05],  # <-- стратегія 2
    [0.05, 0.03, 0.06]], # <-- стратегія 3
)

I_strategies = np.array([5, 4, 5])

P_strategies = [P1, P2, P3]
expected_val = [0, 0, 0]
for i in range(len(P_strategies)):
    expected_val[i] = evaluate(P_strategies[i],
mu_strategies[i], I_strategies[i])

print("За критерієм керування маємо такі очікувані
стратегії: ", expected_val)
print("З них обираємо найкращу (найменшу - найшвидшу,
за введеним критерієм): ", min(expected_val))
P = P_strategies[expected_val.index(min(expected_val))]
mu =
mu_strategies[expected_val.index(min(expected_val))]
I = I_strategies[expected_val.index(min(expected_val))]

print("Тоді за даних умов: ")
print("Швидкість обслуговування в вузлі j: ", mu)
print("Загальна кількість користувачів: ", I)
print("Експоненційний розподіл, задає час обслуговування
в залежності від кількості користувачів в системі у момент
```

```
часу: ", alpha)
print("\nОбираємо стратегію №",
expected_val.index(min(expected_val)) + 1)
print("Та відповідну їй матрицю Маркова: \n", P)
print("\nСимуляція мережі заданою за цією стратегією: ")

async def main():
    f1 = loop.create_task(arrival())
    f2 = loop.create_task(server1())
    f3 = loop.create_task(server2())
    f4 = loop.create_task(server3())
    await asyncio.wait([f1, f2, f3, f4])

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
loop.close()
```

6 Висновки

В роботі розглянуті теоретичні аспекти мереж Джексона та Ґордона - Ньювела, Теорія масового обслуговування. Практичною частиною роботи була розробка програми що симулює мережу Ґордона - Ньювела у реальному часі, та виконує керування. Порівняні знайдене математичне сподівання певної моделі, та її реальне середнє арифметичне значення. За допомогою запропонованого алгоритму можна виконувати керування в мережі Ґордона - Ньювела, а саме за запропонованим критерієм керування (швидкість обслуговування бази клієнтів).

7 Список літератури

1. Gordon, W. J.; Newell, G. F. (1967). “Closed Queuing Systems with Exponential Servers” // P. 254-265.

2. Чорней Р. К. “Дисертація. Локальне керування в мережах” // с. 194-201.

3. Jeffrey P. Buzen Harvard University and Honeywell Information Systems (1973) “Computational Algorithms for Closed Queueing Networks with Exponential Servers”

4. Gong, Q.; Lai, K. K.; Wang, S. Supply chain networks: Closed Jackson network models. and properties. // Vol. 2

5. Queuing theory: Definition, history & real-life applications. (queue-it.com)

6. https://en.wikipedia.org/wiki/Jackson_network

7. https://en.wikipedia.org/wiki/Gordon%E2%80%93Newell_theorem

8. https://en.wikipedia.org/wiki/Rental_utilization