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

**Курсова робота**

освітній ступінь – бакалавр

на тему: **«Керовані марковські поля та їх застосування в економіці»**

Виконав: студент 3-го року навчання
освітньої програми «Комп'ютерні науки»,
спеціальності 122 Комп'ютерні науки

Жидок Федір Андрійович

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

Курсова робота захищена
з оцінкою _____

Секретар ЕК_____
(підпис)

«_____» _____ 20___р.

Київ - 2023

# Зміст

# 1 Вступ

## 1.1 Актуальність

Керовані марковські поля (КМП) є потужним математичним інструментом для моделювання та аналізу різних процесів. Вони дозволяють описувати стохастичні процеси з урахуванням впливу випадкових факторів та рішень, які приймає людина або комп'ютерна програма.

КМП є розширенням теорії марковських процесів, які зазвичай використовуються для аналізу систем з управлінням. Найголовніша особливість в КМП в порівнянні з конвенційними марковським процесом це впровадження механізму прийняття рішень для кожного елементу поля в кожній одиниці часу. Внаслідок цього, наступний стан КМП залежить не тільки від поточного стану, але і від рішень, які приймає кожний елемент в поточній одиниці часу. Прийняття рішень в кожному елементі залежить від поточного стану системи, включаючи стан самого елементу, в якому приймається рішення.

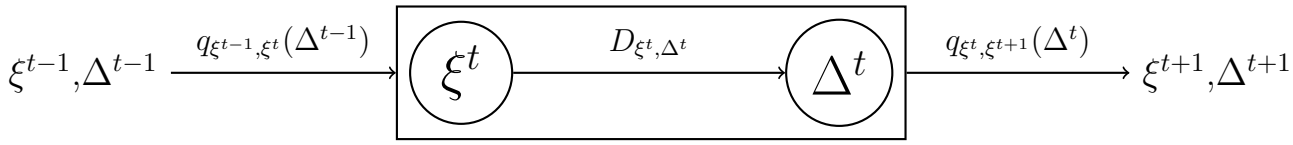


Рис. 1: Схема роботи КМП. $\xi^t$ - випадкова величина, яка відображає стан системи в момент часу $t$. $\Delta^t$ - випадкова величина, яка відображає прийняті рішення в системі в момент часу $t$. $q_{\xi^{t-1}=y,\xi^t=x}(k)$ - ймовірність того, що відбудеться перехід системи в стан $x$, при тому що поточний стан системи є $y$ та прийняті рішення є $k$. $D_{\xi^t=y,\Delta^t=k}$ - ймовірність того, що будуть прийняті рішення $k$, коли стан системи дорівнює $y$.

Керовані марковські поля (КМП) можуть бути застосовані в багатьох галузях економіки, включаючи фінансовий аналіз, управління ризиками та інвестиційні стратегії. Одним з прикладів використання КМП в економіці є моделювання портфеля інвестицій.

Допустимо, що інвестор має портфель, який складається з акцій двох компаній. Кожна компанія може перейти в один з трьох станів: зростання, стабільності або падіння. Інвестор може приймати рішення про купівлю, продаж

або утримання акцій кожної компанії.

Для розв'язання цієї задачі можна використовувати КМП, де стани компаній та рішення інвестора є станами та діями відповідно. Залежно від того, який стан переходить в який після прийняття рішення, розраховується функція виграшу. Інвестор може використовувати КМП для визначення оптимальної стратегії інвестування в залежності від ймовірності переходів компаній між станами та ризиків, пов'язаних з купівлею або продажем акцій.

Інший приклад застосування КМП в економіці полягає в моделюванні процесу ціноутворення на ринку. КМП можуть допомогти управляти динамікою цін на ринку шляхом визначення оптимальної стратегії продажу або купівлі товарів, в залежності від ризиків та ймовірності змін цін.

КМП є потужним інструментом, але для того, щоб його використати, потрібно декілька передумов. По-перше, потрібно мати ймовірності переходів між станами в залежності від прийнятих рішень $\Delta$ та поточного стану $\xi$. По-друге, треба визначити керування для того, щоб рішення, які приймаються в кожний момент часу, були оптимальні. Оптимальність визначається певною функцією витрат, параметрами якої є поточний стан та поточні рішення. Оптимальне керування повинно мінімізовувати функцію витрат, тобто щоб в кожен момент часу приймалося таке рішення, яке мало б найменші витрати або максимізувало прибуток.

## 1.2 Мета дослідження та постановка задачі

Мета цієї курсової роботи є дослідженням проблеми знаходження оптимального керування при відомих ймовірностях $q_{\xi^{t-1},\xi^t}(\Delta^{t-1})$. Об'єктом дослідження є КМП та предметом дослідження є глобальне та локальне керування. Робота складається з трьох розділів.

Перший розділ присвячено формальній постановці задачі, визначення характеристик оптимального керування та визначення задачі в контексті лінійного програмування.

В другому розділі буде досліджено недоліки та переваги задачі знаходження оптимального керування та перехід від початкової глобальної задачі до локальної задля зменшення складності.

Третій розділ буде присвячено визначенню експериментів для порівняння глобальних та локальних керувань, дослідженню результатів та визначенню переваг та недоліків обох стратегій.

В результаті цієї курсової роботи, створено код мовою програмування Python задля вирішення задачі знаходження глобального оптимального керування, локального оптимального керування та проведення експериментів для порівняння глобального та локального керувань на різних випадкових полях.

Постановка задачі:

1. Виконати аналіз задачі знаходження оптимального керування.

2. Дослідити обмеження початкової задачі, яка знаходить глобальне керування.

3. Зробити перехід від задачі по знаходженню глобального керування до задачі по знаходженню локального керування для кожного елементу.

4. Виконати порівняльний аналіз двох видів керувань та зробити висновки стосовно їх застосування. ;

# 2 Аналіз задачі знаходження оптимального керування

## 2.1 Формулювання задачі

Для того, щоб сформулювати задачу про знаходження оптимального керування, треба визначити випадкове поле, марковську властивість в контексті випадкового поля, керування, ймовірнісні позначення, функцію витрат та оптимальність керування. Визначення випадкового поля, керування, властивості Маркова для випадкового поля та керувань, ймовірнісні позначення, оптимальність керування та постановка задачі взяті з [1]. Додатково додана інтерпретація та уточнення деяких вищевказаних елементів для формулювання задачі.

### 2.1.1 Випадкове поле

Випадкове поле [1] являє собою набір випадкових величин, які взаємодіють певним чином одна з одним. Випадкове поле часто репрезентують графом та визначають за допомогою пари $(V, B)$, де $V$ - множина вершин та $B$ - множина ребер. Кожна з вершин є дискретною випадковою величиною, стани або ж область значень для кожної вершини $i$ є $X_i = \{x_i^1, ..., x_i^{n_i}\}$. Випадкова величина на всіх вершинах називається випадковим полем, позначається як $\xi$, та всі стани якого позначаються як $X = \prod_{i \in V} X_i$. Окрім цього, в рамках випадкового поля є поняття околу деякої вершини $i$, яке позначають як $N(i) = \{j : (i, j) \in B\}$ або ж як $\bar{N}(i) = \{j : (i, j) \in B\}$ якщо $i \in N(i)$. Оскільки, в цьому випадку розглядається дискретний випадковий процес, який позначається як $\xi^t$ - значення системи в час t. $\xi_i^t$ є значення вершини $i$ в час $t$.

### 2.1.2 Властивість Маркова на випадковому полі

В рамках випадкового процесу на випадковому полі можна визначити властивість Маркова [1]. Конвенційно, марковський процес є такий випадковий процес, для якого правдиве наступне:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, ..., X_0 = x_0) = P(X_n = x_n | X_{n-1} = x_{n-1})$$

Тобто, поточний стан залежить суто від попереднього і не залежить від більш віддалених станів. Для випадкового поля марковська властивість працює так само, тобто $P(\xi^n = x^n | \xi^{n-1} = x^{n-1}, ..., \xi^0 = x^0) = P(\xi^n = x^n | \xi^{n-1} = x^{n-1})$. Але, розбиваючи цю властивість до кожного окремого елементу системи, в рамках випадкового поля отримується наступне:

$$P(\xi_i^n = x_i^n | \xi^{n-1} = x^{n-1}, ..., \xi^0 = x^0) = P(\xi_i^n = x_i^n | \xi_{\bar{N}(i)}^{n-1} = x_{\bar{N}(i)}^{n-1})$$

До початкової властивості додається залежність на стан зв'язаних з вершиною інших вершин випадкового поля.

### 2.1.3 Визначення керування

Нехай для кожної окремої вершини $i$ визначена деяка множина рішень $U_i$ розмірністю $K$, які можуть прийматися в кожній одиниці часу $t$, та для всі-

єї системи нехай сукупність рішень кожної вершини буде позначатись як $U = \times_{i \in V} U_i$. В теорії, рішення в кожній вершині $i$ в час $t$ приймаються на основі станів $\xi^t_{\bar{N}(i)}$ всіх вершин в околі $i$ в усі одиниці часу $t = 0, ..., T$. Тобто керування [1] в час $t$ для вершини $i$ можна представити як функцію $\Delta^t_i : x_{\bar{N}(i)} \to U_i$, тому просто керування $\delta$ для вершини $i$ дорівнює $\{\Delta^t_i : t \geq 0\}$. Керування може бути марковським, однорідним та допустимим. Керування $\delta_i$ називається марковським [1], якщо $\Delta^t_i(\xi^0_{\bar{N}(i)}, ..., \xi^{t-1}_{\bar{N}(i)}) = \Delta^t_i(\xi^{t-1}_{\bar{N}(i)})$, тобто залежить лише від попереднього стану системи. Однорідне марковське керування [1] не має залежності від $t$ і. е. $\Delta^{t'}_i(X_{\bar{N}(i)}) = \Delta^{t''}_i(X_{\bar{N}(i)}) \forall t', t'' = 0, ..., T$. $\delta_i$ є допустимим [1], якщо $\forall t = 0, ..., T$ всі можливі значення, які може приймати $\Delta^t_i, \subseteq U_i$. Множина таких керувань позначається як $D'$. Окремо важливо позначити множину керувань $D''$ [1], для яких $\forall t = 0, ..., T$ всі можливі значення, які може приймати $\Delta^t_i \equiv U_i$.

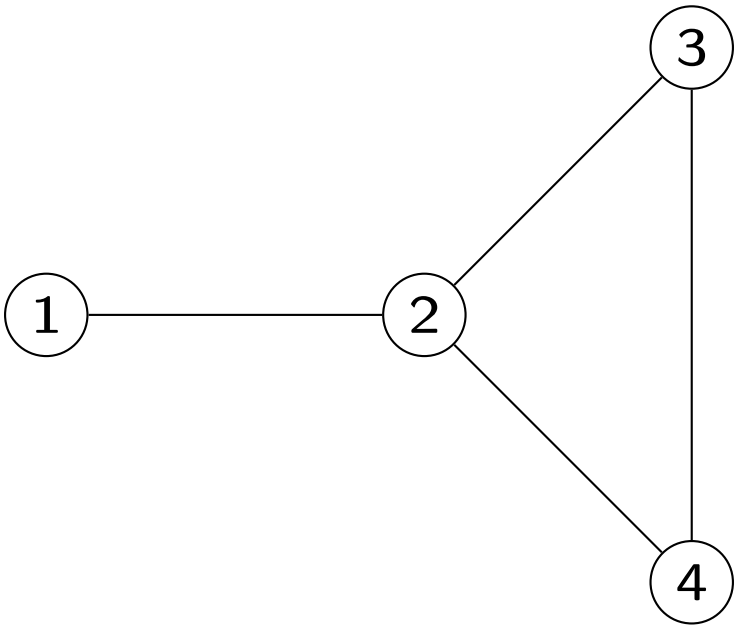


Рис. 2: Приклад випадкового поля. $X_1 = \{1, 2, 3\}$, $X_2 = \{1, 2, 3, 4\}$, $X_3 = X_4 = \{1, 2\}$. $U_j = \{1, 2\} \forall j = 1, ..4$. $\xi^t = [x^t_1, x^t_2, x^t_3, x^t_4]$.

### 2.1.4 Ймовірнісні позначення

В рамках випадкового поля треба визначити декілька ймовірнісних позначень. За схемою 1 роботи КМП треба визначити ймовірність переходу між станами у часі при заданих рішеннях, ймовірність рішення при заданому стані та відособлена ймовірність переходу між станами.

Розглядаючи випадкове поле з властивістю Маркова, ймовірність переходу між станами всієї системи в залежності від рішення позначається як $q_{yx}(k)$ [1], де $y$ - поточний стан системи, $k$ - прийняті рішення в системі зі стану $\xi^t$, $x$ - наступний стан системи.

$$P(\xi^{t+1}|\xi^0, \Delta^0, ..., \xi^t, \Delta^t) \overset{\text{Markovian property}}{\equiv} P(\xi^{t+1}|\xi^t, \Delta^t) \Rightarrow$$

$$\Rightarrow P(\xi^{t+1} = x|\xi^t = y, \Delta^t = k) = q_{yx}(k)$$

Переходи для кожної вершини окремо не залежать один від одного, тобто:

$$P(\xi^{t+1} = x|\xi^t = y, \Delta^t = k) = \prod_{i \in V} P(\xi_i^{t+1} = x_i|\xi_i^t = y_i, \Delta_i^t = k_i)$$

$$= \prod_{i \in V} q_{y_i x_i}(k_i)$$

За аксіомою нормування ймовірності $\sum_{x \in X} P(\xi^{t+1} = x|\xi^t = y, \Delta^t = k) = 1$.

Розглядаючи випадкове поле з властивістю Маркова, ймовірність прийняття рішення при заданому стані всієї системи позначається як $D_{xk}$ [1], де $x$ - поточний стан системи, $k$ - прийняті рішення в системі зі стану $\xi^t = x$.

$$P(\Delta^t|\xi^0, \Delta^0, ..., \xi^t) \overset{\text{Markovian property}}{\equiv} P(\Delta^t|\xi^t) \Rightarrow$$

$$\Rightarrow P(\Delta^t = k|\xi^t = x) = D_{xk}$$

За аксіомою нормування ймовірності $\sum_{k \in U} P(\Delta^t = k|\xi^t = x) = 1$.

Розглядаючи випадкове поле, відособлена ймовірність переходу між станами позначається як $P_{yx}$ [1], де $y$ - поточний стан системи, $x$ - наступний стан системи. Використавши властивість Маркова, $P_{yx}$ можна розкласти наступним чином:

$$P_{yx} = P(\xi^{t+1} = x|\xi^t = y)$$

$$= \sum_{k \in U} P(\xi^{t+1} = x, \Delta^t = k|\xi^t = y)$$

$$= \sum_{k \in U} P(\xi^{t+1} = x|\xi^t = y, \Delta^t = k) * P(\Delta^t = k|\xi^t = y)$$

$$= \sum_{k \in U} q_{yx}(k) * D_{yk}$$

Окрім цих трьох основних позначень, треба ввести позначення $\varphi_{T_{xk}}(y)$ [1], яке визначає частку часу в якій система дорівнює $x$ та приймається рішення $k$.

$$\varphi_{T_{xk}}(y) = \frac{1}{T+1} * \sum_{0}^{T} P(\xi^t = x, \delta^t = k | \xi^0 = y)$$

Позначення $q_{yx}(k)$, $D_{yk}$, $P_{yx}$ та $\varphi_{T_{xk}}(y)$ будуть надалі використовуватись в формулюванні теореми про оптимальне керування, в задачі лінійного програмування в глобальному та локальному контексті.

### 2.1.5 Функція витрат та оптимальність керування

Функція витрат [2] в контексті знаходження оптимального керування описує витрати або ж ціну пов'язану із прийнятими рішеннями та знаходженням в певному стані. Знання даної функції дозволяє знайти оптимальне керування за системою, тобто таке керування, яке забезпечить найкращу ефективність при найменших витратах. Приклад функції витрат для випадкового поля з трьома вершинами з [2]:

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

$$c = [1, 2, 3], b = [3, 1, 2]$$

де $x_j$ - стан вершини $j$ та $u_j$ - рішення вершини $j$. $c$ та $b$ є внутрішні параметри та надалі вони будуть вважатися визначеними.

Оскільки в даній задачі процес є випадковим, то для оцінки певної стратегії треба розглянути очікувану середні витрати за крок. Якщо робити оцінку емпірично, проводячи експерименти з багатьох ітерацій у КМП, то результати будуть відрізнятись та не будуть достатньо репрезентативні. Водночас очікуване середнє буде мати меншу дисперсію як міра того, на скільки витратна та ефективна є стратегія.

Нехай $W^t_{xk}(i)$ - витрата у вершині $i$ при стані $x$, рішенні $k$ в час $t$. Тоді очікувана середня витрата [1] за крок за час $T$ для вершини $i$ визначається наступним чином:

$$Q_{T_i}(y) = \frac{\sum_{t=0,\dots,T} \sum_{x \in X} \sum_{k \in U_i} W^t_{xk}(i) * P(\xi^t = x, \Delta^t_i = k | \xi^0 = y)}{T+1}$$

$$= E[\frac{\sum_{t=0,...,T} W_{xk}^t(i)}{T+1}|\xi_0 = y] = \frac{\sum_{t=0,...,T} E[W_{xk}^t(i)|\xi_0 = y]}{T+1}$$

Потрібно додати, що $T \longrightarrow \infty$, тому що чим більше ітерацій в часі, тим менша дисперсія оцінки середньої ціни за крок. Склавши ці умови, можна формально сформулювати задачу [1] та вивести оптимальність для керування $\delta_i \forall i \in V$.

**Задача**: $\forall i \in V$ треба знайти таке керування $\delta_i$, щоб мінімізовувалась наступна функція:

$$R_y^{\delta_i} = \lim_{T \longrightarrow \infty} \sup \frac{1}{T+1} * \sum_{t=0,...,T} E[W_{xk}^t(i)|\xi_0 = y]$$

Тобто, $\delta_i^*$ є оптимальним [1], якщо $\forall y \in X : \inf_{\delta_i \in D'} R_y^{\delta_i} > R_y^{\delta_i^*}$

## 2.2 Теорема про оптимальне керування та задача лінійного програмування

### 2.2.1 Теорема про оптимальне керування

Перед тим, як прийти до задачі пошуку такого керування, потрібно визначити, чи існує взагалі таке оптимальне керування, для яких випадків та які в нього характеристики. Теорема про існування оптимального керування інтерпретована з [1] та є наступною:

**Теорема**: Якщо простори $X$ і $U_i$ складаються зі скінченної кількості точок, $0 < W_{xk}^t(i) < C < \infty$, то існує оптимальне, однорідне, марковське керування в $D''$ таке, що:

$$\lim_{T \longrightarrow \infty} \varphi_{T_{xk}}(y) = \pi_x * D_{xk}$$

де $\pi_x$ - ергодичні ймовірності $P(\xi = x)$, які задовольняють наступним умовам:

$$\sum_{y \in X} \pi_y P_{yx} = \pi_x, \sum_{y \in X} \pi_y = 1$$

В [1] наведено доведення цієї теореми та в рамках цієї курсової роботи воно не буде розглянуто. Наслідки цієї теореми дозволяють звести задачу з пошуку оптимального керування до задачі лінійного програмування.

### 2.2.2 Задача лінійного програмування

Зведення до проблеми мінімізації з теореми про існування оптимального контролю було взято та інтерпретовано з [1]. Початково, ми маємо $q_{yx}(k)$. Допустимо, що в нас є оптимальне, однорідне, марковське керування $\delta_i^*$. Розглянемо:

$$R_y^{\delta_i^*} = \lim_{T\longrightarrow\infty} \sup \frac{1}{T+1} * \sum_{t=0,\dots,T} E[W_{xk}^t(i)|\xi_0 = y]$$

$$= \lim_{T\longrightarrow\infty} \sup \frac{1}{T+1} * \sum_{t=0,\dots,T} \sum_{x\in X} \sum_{k\in U_i} W_{xk}^t(i) * P(\xi^t = x, \Delta_i^t = k|\xi^0 = y)$$

$$\overset{\text{Homogeneous property}}{\equiv} \lim_{T\longrightarrow\infty} \sup \sum_{x\in X} \sum_{k\in U_i} W_{xk}(i) * \sum_{t=0,\dots,T} P(\xi^t = x, \Delta_i^t = k|\xi^0 = y)$$

$$= \lim_{T\longrightarrow\infty} \sup \sum_{x\in X} \sum_{k\in U_i} W_{xk}(i) * \varphi_{T_{xk}}(y)$$

Теорема стверджує, що при керуванні $\delta_i^*$, отримується наступне $\lim_{T\longrightarrow\infty} \varphi_{T_{xk}}(y) = \pi_x * D_{xk}$. Можна зробити заміну, і ліміт зникне, оскільки ніщо у виразі не буде залежати від часу:

$$= \sup \sum_{x\in X} \sum_{k\in U_i} W_{xk}(i) * \pi_x * D_{xk}$$

В умовах однорідного керування та однорідного марковського поля $\pi_x * D_{xk} = P(\xi = x) * P(\Delta_i = k|\xi = x) = P(\xi = x, \Delta_i = k)$. Оскільки $R_y^{\delta_i^*}$ було необхідно для того, щоб сформулювати теорему та довести її, з більш практичних міркувань ми можемо подивитися на очікувану середню вартість і зробити подібне перетворення, як для $R_y^{\delta_i^*}$:

$$Q_{T_i}(y) = \sum_{x\in X} \sum_{k\in U_i} W_{xk}(i) * \pi_x * D_{xk}$$

За визначенням, оптимальне керування повинно мінімізовувати $Q_{T_i}(y)$. Розглянемо заміну $\pi_x * D_{xk} = z_{xk}$. Тепер спільні ймовірності $z_{xk}$ є параметрами, які можна оцінити, і з них ергодичні ймовірності $\pi_x$ та ймовірності керування $D_{xk}$ може бути отримано наступним чином:

$$\pi_x = \sum_{k\in U_i} z_{xk}, D_{xk} = \frac{z_{xk}}{\pi_x}$$

До того ж необхідно розглянути умови, які були в теоремі. Перш за все, за аксіомою нормованих ймовірностей, $\sum_{x\in X}\sum_{k\in U_i} z_{xk} = 1$ що є еквівалентним умові $\sum_{y\in X}\pi_y = 1$. Остання умова є $\sum_{y\in X}\pi_y P_{yx} = \pi_x$ і для того, щоб її виконати, треба вивести її через $z_{xk}$ та $q_{yx}(k)$:

$$\forall x \in X : \sum_{y\in X}\pi_y P_{yx} = \pi_x$$

$$\sum_{y\in X} P(\xi^t = y, \xi^{t+1} = x) = \sum_{k\in U_i} z_{xk}$$

$$\sum_{y\in X}\sum_{k\in U_i} P(\xi^t = y, \Delta_i^t = k) * P(\xi^{t+1} = x | \xi^t = y, \Delta_i^t = k) = \sum_{k\in U_i} z_{xk}$$

$$\sum_{y\in X}\sum_{k\in U_i} z_{yk} * q_{yx}(k) = \sum_{k\in U_i} z_{xk}$$

Оскільки ця умова виражається лише за допомогою відомих змінних і цільових параметрів, ми можемо додати її до задачі. Однією з найважливіших умов є те, що $z_{xk}$ повинні бути позитивними оскільки ймовірності не можуть бути негативні. Також важливо відзначити, що у формулюваннях і теоремі керування $\delta$ розглядалося лише для однієї вершини випадкового поля, але всі наведені вище поняття можна розширити до $\delta$ на всьому випадковому полі. Простіше працювати з $\delta$ на всьому випадковому полі в контексті функції вартості. Нарешті, проблему мінімізації [1] можна сформулювати, оскільки всі умови були враховані:

$$\begin{gathered}
\underset{x}{\text{minimize}} \sum_{x\in X}\sum_{k\in U} W_{xk} * z_{xk} \\
\sum_{y\in X}\sum_{k\in U} z_{yk} * q_{yx}(k) = \sum_{k\in U} z_{xk} \ \forall x \in X \\
\sum_{x\in X}\sum_{k\in U} z_{xk} = 1 \\
z_{xk} \geq 0,\ x \in X, k \in U
\end{gathered}$$

;

# 3 Глобальне та локальне керування

## 3.1 Приклад визначення глобального керування за допомогою $D_{xk}$

Сформульована задача лінійного програмування знаходить оптимальне керування через $D_{xk}$. За $D_{xk}$ певний стан системи призводить до певного набору рішень у кожному вузлі. Зазвичай $D_{xk}$ є розрідженим, тому легко відстежити, які рішення є найвигіднішими для кожного стану. У додатковому матеріалі [2] є приклад циклічної системи черги з визначеним $q_{yx}(k)$. В цілому система представлена наступним чином: 3.

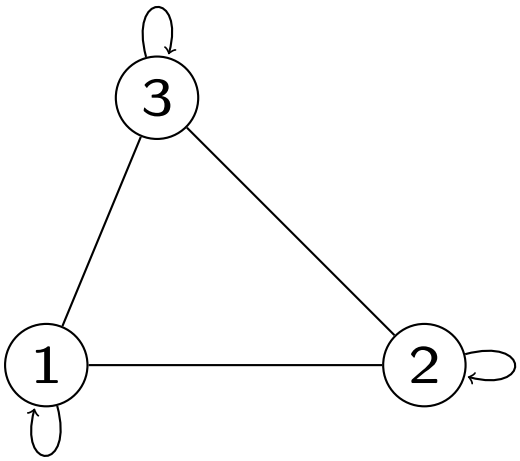


Рис. 3: Випадкове поле циклічної черги.

$$X_j = \{0, 1, 2\}, U_j = \{0, 1\} \forall j = 1, 2, 3.$$

$$X = \{(0, 0, 2), (2, 0, 0), (0, 2, 0), (1, 1, 0), (0, 1, 1), (1, 0, 1)\}.$$

$$U = \{(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1)\}$$

У цьому випадку розглядалися лише вибрані значення системи $X$, оскільки система репрезентує циклічну чергу і не може приймати деякі конкретні значення. Після розв'язання цієї задачі лінійного програмування за допомогою симплексного методу отримано матрицю $z_{xk}$ розмірності шість на вісім.

В такому випадку, з матриці керування 1 для кожного стану $x_i$ відповідає набір рішень $u_i$. Ця задача має більш глобальний контекст, тобто стани та рішення визначались для всієї системи одночасно, керування для окремих вершин не розглядалося.

| | **(0, 0, 0)** | **(0, 0, 1)** | **(0, 1, 0)** | **(0, 1, 1)** | **(1, 0, 0)** | **(1, 0, 1)** | **(1, 1, 0)** |
|---|---|---|---|---|---|---|---|
| (0, 0, 2) | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| (0, 1, 1) | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| (0, 2, 0) | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| (1, 0, 1) | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| (1, 1, 0) | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| (2, 0, 0) | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Табл. 1: $D_{xk}$ - control matrix

Розв'язанням задачі пошуку оптимального керування дивлячись на всю систему загалом в наступному буде називатись пошуком глобального керування.

## 3.2 Обмеження глобального керування

В прикладі 3 ми знаходили глобальне розв'язання задачі, а саме ми не розглядали окремо керування для кожної вершини, а знаходили керування для всієї системи одночасно. Такий підхід до розв'язання задачі пошуку оптимального керування має свої переваги та недоліки.

Перевагою глобального керування є те, що набагато легше використовувати саме глобальну стратегію, бо розв'язуючи одну задачу утворюється універсальний розв'язок для всієї системи, виведений за допомогою урахування всіх можливих варіантів. Але водночас це є і недоліком глобального розв'язку, бо в такому випадку задача має велику часову складність та при розширенні випадкового поля дуже швидко стає складно розв'язати цю задачу.

Наприклад, розв'язати задачу по знаходженню глобального оптимального контролю для випадкового поля 4 вже не є можливим, бо кількість параметрів сягає більше шестидесяти тисяч, хоча в даному випадку представлене випадкове поле лише з шести вершин. Глобальне оптимальне керування буде дуже складно знайти для великих випадкових полів, тому потрібно розглянути інші варіанти.

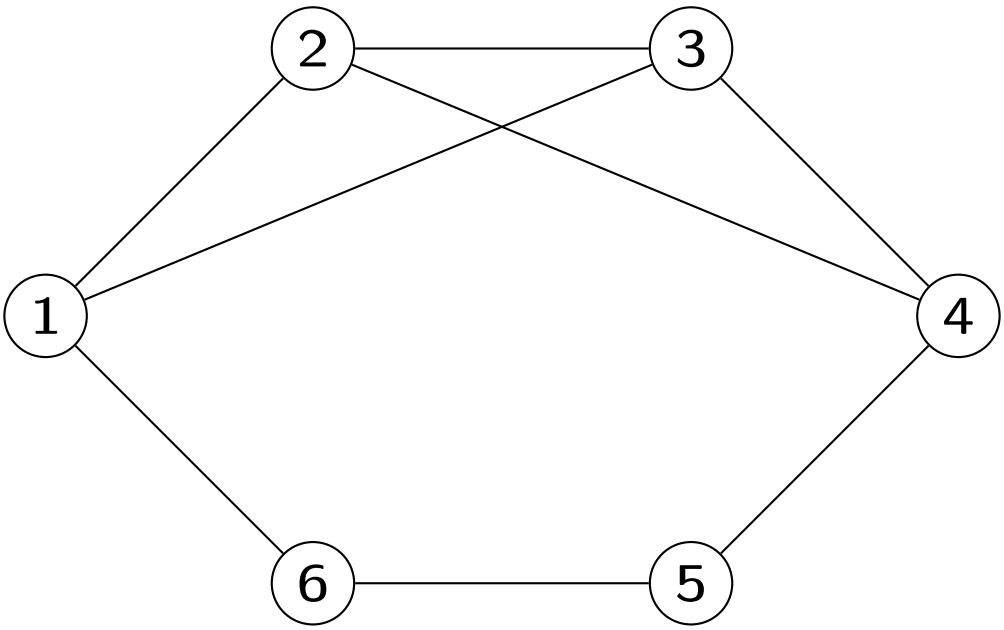


Рис. 4: Випадкове поле, для якого вже неможливо визначити глобальне керування. $X_j = \{0, 1, 2\} \forall j = 1, \ldots, 5, X_6 = \{0, 1, 2, 3\}$.
$U_j = \{0, 1\} \forall j = 1, \ldots, 6$

Окрім проблем зі складністю, глобальне керування не можна розкласти в більш локальні випадки, тобто розглядати окремо керування для кожної вершини. Саме тому треба зробити декомпозицію глобальної, великої задачі на декілька локальних, тим самим набагато зменшивши розмірність кожної з локальних задач та отримавши локальні керування, які можна буде об'єднати в глобальне керування.

## 3.3 Перехід до задачі локального керування

Для того, щоб виконати перехід до локального керування кожної окремої вершини $i \in V$, треба адаптувати початкову задачу лінійного програмування та використати властивості оптимального керування з теореми про існування оптимального керування.

### 3.3.1 Використання властивості Маркова

Оскільки треба адаптувати задачу під локальне керування, то можна це зробити через властивість Маркова, а саме $P(\xi_i^{t+1} = x | \xi^t = y, \Delta_i^t = k) \equiv P(\xi_i^{t+1} = x | \xi_{\bar{N}(i)}^t = y_{\bar{N}(i)}, \Delta_i^t = k)$. В рамках локального оптимального керування, розглянемо $z_{xk} = P(\xi_i^t = x, \Delta_i^t = k)$. Тоді функцію очікуваних витрат за

крок для мінімізації можна представити як:

$$\sum_{x\in X_i}\sum_{k\in U_i} W_{xk}(i) * z_{xk}$$

Обмеження $\sum_{x\in X_i}\sum_{k\in U_i} z_{xk} = 1$ та $z_{xk} \geq 0,\ x \in X_i, k \in U_i$ зберігаються.

### 3.3.2 Адаптація обмеження про перехідні ймовірності

Залишається адаптувати останнє обмеження:

$$\sum_{y\in X}\sum_{k\in U} z_{yk} * q_{yx}(k) = \sum_{k\in U} z_{xk}, \quad \forall x \in X$$

Достатньо його просто переписати в контексті умови зазначеній в теоремі про оптимальне керування:

$$\forall x \in X : \sum_{y\in X} \pi_y P_{yx} = \pi_x \quad \Longrightarrow \quad \forall x \in X_i : \sum_{y\in X_i} \pi_y P_{yx} = \pi_x$$

$$\sum_{y\in X_i} P(\xi_i^t = y, \xi_i^{t+1} = x) = \sum_{k\in U_i} z_{xk}$$

$$\sum_{y\in X_i}\sum_{k\in U_i} P(\xi_i^t = y, \Delta_i^t = k) * P(\xi_i^{t+1} = x | \xi_i^t = y, \Delta_i^t = k) = \sum_{k\in U_i} z_{xk}$$

$$\sum_{y\in X_{\bar{N}(i)}}\sum_{k\in U_i} P(\xi_{\bar{N}(i)}^t = y, \Delta_i^t = k) * P(\xi_i^{t+1} = x | \xi_{\bar{N}(i)}^t = y, \Delta_i^t = k) = \sum_{k\in U_i} z_{xk}$$

Зміна $\xi_t$ на $\xi_{\bar{N}(i)}^t$ дає можливість розглядати окіл тільки для вершини $i$ на випадковому полі і за властивістю Маркова цей перехід можливо зробити та не втратити точність.

$$\sum_{y\in X_{\bar{N}(i)}}\sum_{k\in U_i} P(\xi_{\bar{N}(i)}^t = y, \Delta_i^t = k) * q_{yx}(k) = \sum_{k\in U_i} z_{xk}$$

Переписавши умову в ймовірнісних позначеннях, видно, що неможливо паралельно визначити $P(\xi_{\bar{N}(i)}^t = y, \Delta_i^t = k)$, тому потрібно зробити деяку оцінку цієї величини зліва, яка б була максимально наближена до справжньої. Переписавши ліву частину умови отримуємо:

$$\sum_{y\in X_{\bar{N}(i)}}\sum_{k\in U_i} P(\xi_{\bar{N}(i)}^t = y, \Delta_i^t = k) * q_{yx}(k) = \mathbb{E}_{\xi_{\bar{N}(i)}^t, \Delta_i^t} q_{yx}(k)$$

З емпіричної точки зору, очікуване значення $q_{yx}(k)$ буде дорівнювати середньому між всіма $q_{yx}(k) \forall y \in X_{\bar{N}(i)}, k \in \xi^t_{\bar{N}(i)}$. Отож була отримана ідентична задача лінійного програмування:

$$\begin{gathered}
\underset{x}{\text{minimize}} \sum_{x \in X_i} \sum_{k \in U_i} W_{xk}(i) * z_{xk} \\
\frac{\sum_{y \in X_{\bar{N}(i)}} \sum_{k \in U_i} q_{yx}(k)}{|X_{\bar{N}(i)}| * |U_i|} = \sum_{k \in U_i} z_{xk}, \quad \forall x \in X_i \\
\sum_{x \in X_i} \sum_{k \in U_i} z_{xk} = 1 \\
z_{xk} \geq 0, \ x \in X_i, k \in U_i
\end{gathered}$$

### 3.3.3 Переваги та недоліки локального керування

Локальне оптимальне керування має свої переваги та недоліки. По-перше, за допомогою задачі локального оптимального керування можна набагато швидше отримати результати для великих випадкових полів, в той час, як глобальний випадок може вирішуватись дуже довго або ж взагалі не буде знайдений. По-друге, переваги локального випадку в тому, що з нього можна утворити глобальний випадок, водночас глобальний випадок неможливо розкласти до кожного елементу випадкового поля. Однак, оскільки для локального випадку довелося робити деяку заміну в одній з умов, то рішення, дане в результаті локального випадку, не буде найоптимальнішим та буде давати гірші результати по мінімізації, ніж в глобальному випадку.

### 3.3.4 Інші варіанти декомпозиції

Також необхідно зазначити, що можна було зробити більш проміжну декомпозицію глобальної задачі та розглядати окремо $z_{xk} = P(\xi^t = x, \Delta^t_i = k)$ замість глобального випадку, де $z_{xk} = P(\xi^t = x, \Delta^t = k)$. Але така декомпозиція не є дуже ефективною, бо при розширенні випадкового поля в вершинах, кількість параметрів буде зростати степенево. Ідеальний варіант декомпозиції міг би бути при $z_{xk} = P(\xi^t_{\bar{N}(i)} = x, \Delta^t_i = k)$, таким чином можна було б залучити Марковську властивість ще більше та отримати:

$$\sum_{y \in X} \sum_{k \in U} z_{yk} * q_{yx}(k) = \sum_{k \in U} z_{xk}, \quad \forall x \in X_{\bar{N}(i)} \Longrightarrow$$

$$\sum_{y \in X_{\bar{N}(i)}} \sum_{k \in U_i} P(\xi^t_{\bar{N}(i)} = y, \Delta^t_i = k) * P(\xi^{t+1}_{\bar{N}(i)} = x | \xi^t_{\bar{N}(i)} = y, \Delta^t_i = k) =$$

$$= \sum_{k \in U_i} P(\xi^t_{\bar{N}(i)} = x, \Delta^t_i = k) \quad \forall x \in X_{\bar{N}(i)}$$

Але обрахунок $P(\xi^{t+1}_{\bar{N}(i)} = x | \xi^t_{\bar{N}(i)} = y, \Delta^t_i = k)$ є нетривіальною задачею та розкласти цей вираз на щось більш здійснене для оцінки/розкладу неможливо.
;

# 4 Емпіричне порівняння глобального та локального керувань

Наразі, для того щоб підтвердити деякі твердження в минулій секції та порівняти глобальну та локальні задачі було виконано ряд експериментів на різних випадкових полях.

## 4.1 Постановка експериментів

Для кожного експерименту визначено випадкове поле, тобто множина вершин $V$, ребер $E$, станів $X_i$ та рішень $U_i$ для кожної вершини $i \in V$. Експериментом вважається наступна послідовність дій:

**1.** Визначення локальної матриці $Q^{\text{local}}_{yx}(k)$ випадковим чином та глобальної матриці $Q^{\text{global}}_{yx}(k)$ використовуючи локальну матрицю. $Q^{\text{local}}_{yx}(k) \sim Dir(.., n_{x_j}, ..)$, де $n_i$ - випадкове число, яке відповідає за параметр для визначення ймовірностей значення $x_j$ в вершині $i$ при попередньому стані сусідів $y$ та рішені $k$.
$Q^{\text{global}}_{yx}(k) =$
$= \prod_{i \in V} Q^{\text{local}}_{y_{\bar{N}(i)} x_i}(k_i)$.

**2.** Знаходження локального та глобального оптимального керувань.

**3.** Переведення локального керування у глобальне задля того, щоб порівняти глобальну очікувану витрату за крок та локальну. Переведення зробити доволі просто, оскільки $P(\xi = x, \Delta = k) = \prod_{i \in V} P(\xi_i = x_i, \Delta_i = k_i)$.

Оцінюватись в такому випадку буде швидкість знаходження кожного з виду керувань та наскільки мала очікувана витрата за кожний крок. Для ефективного порівняння було використано декілька різних випадкових полів та для кожного з них експеримент проводився десять разів. Випадкові поля, використані для експериментів:

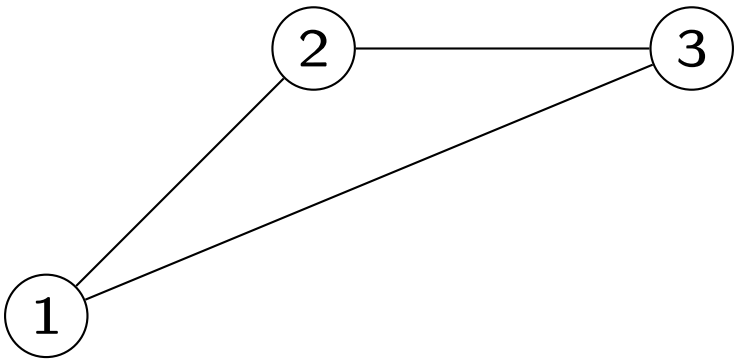


Рис. 5: 3 вершини з максимальною кількістю ребер.
$V = (1, 2, 3), E = ((1, 2), (2, 3), (1, 3))$. $\forall i \in V : X_i = (1, 2, 3), U_i = (1, 2)$

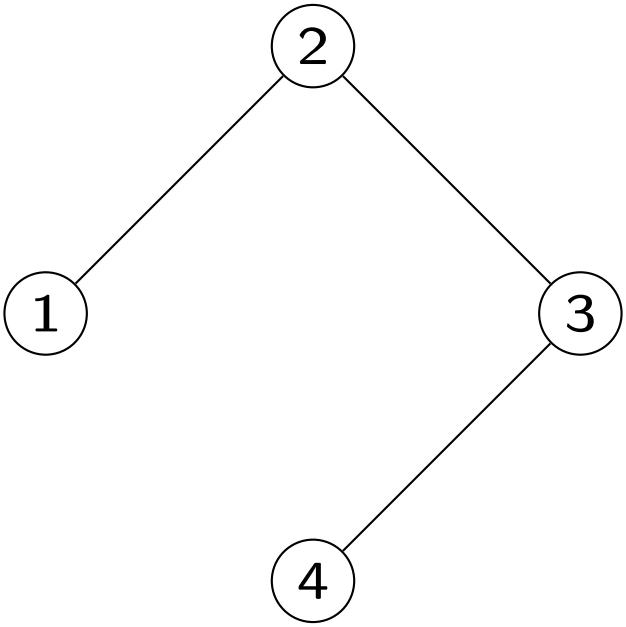


Рис. 6: 4 вершини, випадкове поле з невеликою кількістю зв'зків.
$V = (1, 2, 3, 4), E = ((1, 2), (2, 3), (3, 4))$. $\forall i \in V : X_i = (1, 2, 3), U_i = (1, 2)$

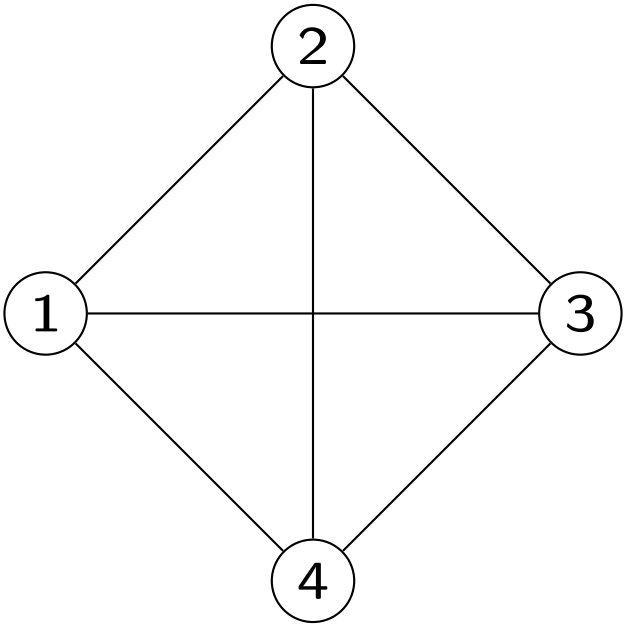

Рис. 7: 4 вершини, випадкове поле з великою кількістю зв'зків.

$V = (1, 2, 3, 4), E = ((1, 2), (2, 3), (3, 4), (1, 3), (1, 4), (2, 4)).$

$\forall i \in V : X_i = (1, 2, 3), U_i = (1, 2)$

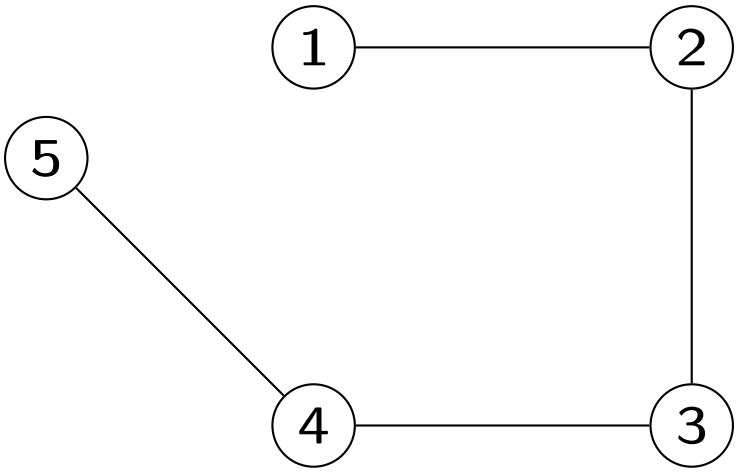


Рис. 8: 5 вершин, випадкове поле з невеликою кількістю зв'зків.

$V = (1, 2, 3, 4, 5), E = ((1, 2), (2, 3), (3, 4), (4, 5)).$

$\forall i \in V : X_i = (1, 2, 3), U_i = (1, 2)$

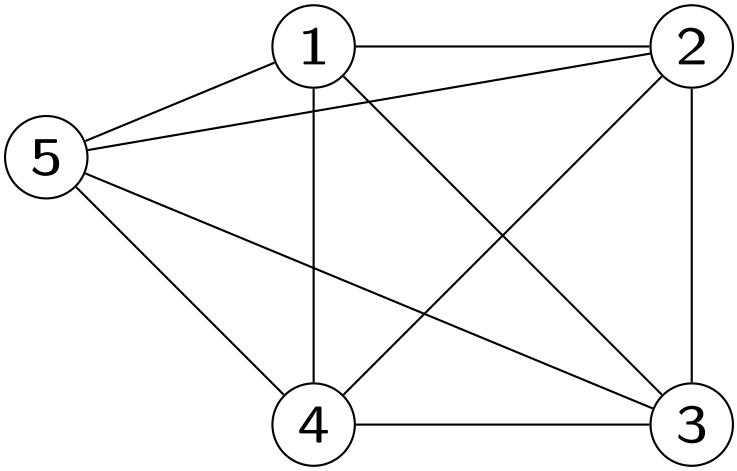


Рис. 9: 5 вершин, випадкове поле з великою кількістю зв'зків.

$V = (1, 2, 3, 4, 5)$

$E = ((1, 2), (2, 3), (3, 4), (4, 5), (1, 3), (1, 4), (1, 5), (2, 4), (2, 5), (3, 5)).$

$\forall i \in V : X_i = (1, 2, 3), U_i = (1, 2)$

Для кращої репрезентативності я додав випадкові поля, які мають багато зв'язків, та ті, які навпаки їх мають мало. Випадкові поля з шістьма та більше вершинами не розглядались, бо пошук глобального керування потребує дуже багато ресурсів.

## 4.2 Аналіз результатів

Табл. 2: Результати експериментів для випадкового поля з трьома вершинами

| Sim. № | Global Strategy Time | Local Strategy Loss | Global Strategy Time | Local Strategy Time |
|---|---|---|---|---|
| 1 | 10.516 | 10.86 | 0.02605 | 0.00747 |
| 2 | 10.456 | 10.61 | 0.025 | 0.007576 |
| 3 | 10.56 | 10.77 | 0.02498 | 0.007366 |
| 4 | 10.516 | 10.68 | 0.02454 | 0.0074 |
| 5 | 10.555 | 10.78 | 0.02592 | 0.00778 |

Результати експериментів 2 підтверджують наведені в минулому секторі твердження стосовно теоретичної різниці локального та глобального керувань. З результатів видно, що глобальне керування краще мінімізує витрати, в середньому на два відсотки менше у відношенні до локального. Тобто якщо брати очікувану ціну за крок у глобальному керуванні як 100 відсотків, то в середньому очікувана ціна за крок у локальному керуванні буде в пропорції складати 102 відсотки. Окрім загальної картини, можна подивитись на окремі значення для різних випадкових полів 3.

Табл. 3: Середні значення відношень у відсотках для кожного з випадкових полів

| Random Field | Local to Global ratio Percentage | Global to Local ratio Percentage |
|---|---|---|
| RandomField 5 | 1.87 | 98.17 |
| RandomField 6 | 2.95 | 97.16 |
| RandomField 7 | 1.84 | 98.19 |
| RandomField 8 | 1.87 | 98.17 |
| RandomField 9 | 1.29 | 98.73 |

Виходить, що в середньому найменша різниця між локальним та глобаль-

ним керуванням була у випадкового поля з п'ятьма вершинами та найбільшою кількістю ребер. В той самий час, найбільш відчутна надбавка витрат за крок у випадкового поля з чотирма вершинами, де різниця сягає в три відсотки.

Табл. 4: Середня кількість часу, який займає пошук кожної зі стратегій

| Random Field | Global strategy time(sec.) | Local strategy time(sec.) |
|---|---|---|
| RandomField 5 | 0.024996 | 0.007485 |
| RandomField 6 | 0.185354 | 0.020206 |
| RandomField 7 | 0.184221 | 0.021178 |
| RandomField 8 | 66.141925 | 0.092156 |
| RandomField 9 | 94.671790 | 0.101406 |

З часових показників 4 очевидно, що локальне керування є набагато більш ефективним в плані часу ніж глобальне. З прогресією за кількістю вершин помітно, що час для пошуку глобальної стратегії змінюється в набагато швидшому темпі, ніж для локальної.

Оскільки аналіз експериментів завершено, можна зробити відповідні підсумки. ;

# 5 Висновки

В даній курсовій роботі була досліджена проблематика знаходження оптимального керування. Було виведено формально задачу, огляд теореми про існування оптимального керування та перенесено цю задачу в контекст лінійного програмування. Вже в цьому контексті, було розглянуто різні варіанти того, як можна знаходити оптимальне керування, було визначено недоліки початкової задачі та проведено її декомпозицію на менші задачі пошуку локального керування. В заключені, було проведено порівняння обох видів керувань та підтверджено деякі теоретичні допущення вище стосовно швидкості знаходження та очікуваних витрат.

Отже, в цій курсовій роботі було проведено аналіз заданої задачі, визначено спосіб для декомпозиції початкової задачі та отриманню потенціально кращих результатів.

# 6 Список літератури

# Література


[1] P. S. Knopov and R. K. Chornei. *Control Problems for Markov Processes with Memory* in Cybernetics and Systems Analysis, Vol. 34, No. 3 (1998);

[2] P. S. Knopov, R. K. Chornei and H. Daduna. *Controlled Markov Fields with Finite State Space on Graphs* in Stochastic Models, Vol. 21, No. 4 (2005);

[3] P. S. Knopov. *Markov fields and their applications in economics* in Journal of Mathematical Sciences, Vol. 97 (1999);

[4] B. Hajek. *Random Processes for Engineers* by Cambridge: Cambridge University Press, pp. 1-177 (2015);

[5] C. Derman. *Finite State Markovian Decision Processes* by Academic Press: New York, pp. 2-50 (1970);


# 7 Додаток

```
class MarkovianRFWithGlobalControl():

    def __init__(self, X: List[Tuple[int]], U: List[List[int]], \\
        Q: defaultdict, C: np.array, B: np.array) -> None:
        self.X = X
        self.U = U
        self.Q = Q
        self.C = C
        self.B = B
```

```
        self.Z = None
        self.minW = None

    def W(self, x: Tuple[int], u: List[int]):
        return np.sum((self.C - np.array(u)) * np.array(x) \\
            + np.array(u) * self.B)

    def find_optimal_control(self):
        res = linprog(c=self.c, A_eq=self.A_eq, b_eq=self.b_eq, \\
            method = 'simplex')
        self.Z = np.reshape(res.x, (-1, len(self.U)))
        self.minW = 0
        for i, x in enumerate(self.X):
            for j, u in enumerate(self.U):
                self.minW += self.Z[i, j] * self.W(x, u)

    def __a_eq(self, x):
        coefs = list()
        for y in self.X:
            if y == x:
                [
                    coefs.append(self.Q[x][x][u] - 1)
                    for u in self.U
                ]
                continue
            for u in self.U:
                coefs.append(self.Q[y][x][u])
        return np.array(coefs)

    def __prob_constr(self):
        return np.array([1 for _ in range(len(self.X) * \\
            len(self.U))])
```

```
    @property
    def b_eq(self):
        return np.array([0 for _ in range(len(self.X))] + [1])

    @property
    def A_eq(self):
        coefs = list()
        for x in self.X:
            coefs.append(self.__a_eq(x))
        coefs.append(self.__prob_constr())
        return np.array(coefs)

    @property
    def c(self):
        coefs = list()
        for x in self.X:
            for u in self.U:
                coefs.append(self.W(x, u))
        return np.array(coefs)

    @property
    def ergodic(self):
        pi = np.sum(self.Z, axis = 1)
        return pd.Series(pi, index = self.X)

    @property
    def control(self):
        D = (self.Z.T / np.sum(self.Z, axis = 1)).T
        return pd.DataFrame(data = D, index = self.X, \\
            columns = self.U)

class MarkovianRFWithLocalControl():

    def __init__(self, V: Tuple[int], Vk: Tuple[int], \\
```

```
        Uk: Tuple[int], E: Tuple[Tuple[int, int]], \\
        Q: defaultdict, C: np.array, B: np.array) -> None:
        self.V = V
        self.Vk = Vk
        self.Uk = Uk
        self.E = E
        self.Q = Q
        self.C = C
        self.B = B
        self.minW = None
        self.Z = None

    def Wlocal(self, v: int, x: int, u: int):
        return (self.C[v] - u) * x + u * self.B[v]

    def W(self, x: Tuple[int], u: List[int]):
        return np.sum((self.C - np.array(u)) * np.array(x) \\
            + np.array(u) * self.B)

    def find_optimal_control(self):
        Zlocal = dict()
        local_strategies = list(product(self.Vk, self.Uk))
        for v in self.V:
            res = linprog(c=self.c(v), A_eq=self.A_eq, \\
                b_eq=self.b_eq(v), method = 'simplex')
            Zlocal[v] = res.x
        U = [*product(*[Uk for _ in range(len(self.V))])]
        X = [*product(*[Vk for _ in range(len(self.V))])]
        Z = list()
        minW = 0
        for x in X:
            Zx = list()
            for u in U:
```

```
                proba = 1
                for v, x_v, u_v in zip(self.V, x, u):
                    i = local_strategies.index((x_v, u_v))
                    proba *= Zlocal[v][i]
                Zx.append(proba)
                minW += proba * self.W(x, u)
            Z.append(np.array(Zx))
        self.Z = np.array(Z)
        self.minW = minW
        self.X = X
        self.U = U

    def c(self, v: int):
        coefs = list()
        for x in self.Vk:
            for u in self.Uk:
                coefs.append(self.Wlocal(v - 1, x, u))
        return np.array(coefs)

    def b_eq(self, v):
        adj_v = [vertex for e in self.E if v in e for vertex \\
            in e if vertex != v]
        Y = [tuple(zip(adj_v, y)) for y in [*product(*[self.Vk \\
            for _ in adj_v])]]
        constraints = list()
        for x in self.Vk:
            mean = sum([self.Q[(v,x)][y][u] for y in Y for u \\
                in self.Uk]) / (len(Y) * len(self.Uk))
            constraints.append(mean)
        return np.array(constraints)

    @property
    def A_eq(self):
```

```
        coefs = list()
        for i, _ in enumerate(self.Vk):
            x_coefs = np.zeros(shape = (len(self.Vk) * \\
                len(self.Uk), ))
            x_coefs[list(range(i * len(self.Uk), (i + 1) * \\
                len(self.Uk)))] = 1
            coefs.append(x_coefs)
        return np.array(coefs)

    @property
    def __prob_constr(self):
        return np.array([1 for _ in range(len(self.Vk) \\
            * len(self.Uk))])

    @property
    def ergodic(self):
        pi = np.sum(self.Z, axis = 1)
        return pd.Series(pi, index = self.X)

    @property
    def control(self):
        D = (self.Z.T / np.sum(self.Z, axis = 1)).T
        D[np.isnan(D)] = 0
        return pd.DataFrame(data = D, index = self.X, \\
            columns = self.U)

class RandomField():

    def __init__(self, V: Tuple[int], Vk: Tuple[int], \\
        Uk: Tuple[int], E: Tuple[Tuple[int, int]], \\
        rand_int_n: int = 100, Qlocal: defaultdict = None, \\
        globally: bool = False):
        self.V = V
        self.Vk = Vk
```

```
        self.Uk = Uk
        self.E = E
        self.globally = globally
        self.rand_int_n = rand_int_n
        self.Qlocal = Qlocal if Qlocal is not None \\
            else self.__generate_local_Q()
        if self.globally:
            self.X = [*product(*[self.Vk for _ in \\
                range(len(self.V))])]
            self.U = [*product(*[self.Uk for _ in \\
                range(len(self.V))])]
            self.Qglobal = self.__generate_global_Q()

    def __generate_local_Q(self):
        Qlocal = defaultdict(lambda: defaultdict(\\
            lambda: defaultdict(int)))
        for v in self.V:
            adj_v = [vertex for e in self.E if v in \\
                e for vertex in e if vertex != v]
            Vs = [tuple(zip(adj_v, y)) for y in \\
                [*product(*[self.Vk for _ in adj_v])]]
            for y in Vs:
                for u in self.Uk:
                    x_prob = np.random.dirichlet(\\
                        np.random.randint(low = 1, \\
                            high = self.rand_int_n, \\
                            size=len(self.Vk)), \\
                            size=1)[0]
                    for i, x in enumerate(Vk):
                        Qlocal[(v,x)][y][u] = x_prob[i]
        return Qlocal

    def __generate_global_Q(self):
```

```
        Qglobal = defaultdict(lambda: defaultdict(\\
            lambda: defaultdict(int)))
        for y in self.X:
            for u in self.U:
                for state in self.X:
                    proba = 1
                    for v, x in zip(self.V, state):
                        adj_v = np.array([vertex for e \\
                            in self.E if v in e for vertex \\
                            in e if vertex != v])
                        y_v = tuple(zip(adj_v, list(\\
                            np.array(y)[adj_v - 1])))
                        proba *= self.Qlocal[(v, x)][y_v][u[v-1]]
                    Qglobal[state][y][u] = proba
        return Qglobal

    def plot(self):
        G = nx.Graph()
        G.add_edges_from(self.E)
        nx.draw(G, with_labels=True)
        plt.show()

    def copy(self, gen_local = True):
        args = self.V, self.Vk, self.Uk, self.E
        kwargs = {"globally": self.globally}
        if not gen_local:
            kwargs = {"Qlocal": self.Qlocal}
        return self.__class__(*args, **kwargs)

    def __str__(self):
        return f"RandomField(V = {self.V}; Vk = {self.Vk}; \\
            Uk = {self.Uk}; E = {self.E})"
```

```
    @property
    def n_of_vertexes(self):
        return len(self.V)

    @property
    def n_of_states(self):
        return len(self.Vk)

    @staticmethod
    def from_string(string: str):
        args = list()
        for var in ["V", "Vk", "Uk"]:
            var_unp, string = string.split(";")[0], \\
                string.split(";")[1:]
            value = eval(var_unp.split(f"{var} = ")[1])
            args.append(value)
        value = eval(var_unp.split(f"E = ")[1][:-1])
        args.append(value)
        return RandomField(*args)

class ComparisonSimulations():


    def __init__(self, random_fields: List[RandomField], \\
        C: np.array, B: np.array, \\
        n_of_sim_for_each_field: int = 20, n_jobs: int = -1):
        self.random_fields = random_fields
        self.C = C
        self.B = B
        self.n_of_sim_for_each_field = n_of_sim_for_each_field
        self.n_jobs = n_jobs
        self.results = list()

    def run_simulations(self):
```

```
parallel = Parallel(n_jobs=-1, pre_dispatch="2*n_jobs",\\
    backend="loky", prefer="processes")

def _fit_and_get_scores(random_field_origin: \\
    RandomField, n_sim: int, C: np.array, \\
    B: np.array, worker_random_state: int):
    import numpy as np
    seed(None)

    random_field = random_field_origin.copy()
    C = C[:random_field.n_of_vertexes]
    B = B[:random_field.n_of_vertexes]

    local_rf = MarkovianRFWithLocalControl( \\
        random_field.V, random_field.Vk, \\
        random_field.Uk, random_field.E, \\
        random_field.Qlocal, C, B)
    start = time.time()
    local_rf.find_optimal_control()
    local_rf_time = time.time() - start

    global_rf = MarkovianRFWithGlobalControl(\\
        random_field.X, random_field.U, \\
        random_field.Qglobal, C, B)
    start = time.time()
    global_rf.find_optimal_control()
    global_rf_time = time.time() - start

    return list([str(random_field), n_sim, \\
        global_rf.minW, local_rf.minW, \\
        global_rf_time, local_rf_time])

self.results = parallel(
```

```
            delayed(_fit_and_get_scores)(random_field, \\
                num_simulation, self.C, self.B, \\
                np.random.randint(low=1, high=1e9))
            for random_field in self.random_fields
            for num_simulation in range(1, \\
                self.n_of_sim_for_each_field + 1)
        )
        return self.results

    @property
    def results_csv(self):
        return pd.DataFrame(data = self.results, \\
            columns = ["Random Field", "Simulation", \\
            "Global strategy Loss", "Local strategy Loss", \\
            "Global strategy Time", "Local strategy Time"])
;
```