КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра моделювання складних систем

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня бакалавра
за спеціальністю 113 «Прикладна математика»

на тему:

Випадкові vp-дерева та пов'язані ланцюги Маркова на просторі замкнених множин

Виконавець:
бакалавр четвертого курсу

Цимбал Софія Анатоліївна

Науковий керівник:
професор, доктор фіз.-мат. наук

Маринич Олександр Віталійович

Роботу заслухано на засіданні кафедри моделювання складних систем та рекомендовано до захисту, протокол №10 від 05.06.2023

Завідувач кафедри МСС
д.т.н., доцент Дмитро Черній

Київ — 2023

Зміст

Анотація

У даній роботі вивчаються властивості метричних дерев, зокрема vp-дерев, та пов'язаних із ними ланцюгів Маркова. Розглядаються теоретичні основи vp-дерев та їх побудова. Використовуючи ланцюги Маркова, досліджуються властивості послідовності розділів для побудови дерева. Робота включає в себе моделювання, в якому vp-дерева будуються на реальних випадково розподілених наборах даних.

1 Огляд літератури

1.1 Дерева сортування та зберігання даних

Визначення 1.1.1 *Дерева пошуку — тип структур даних, який використовується для ефективного пошуку, вставки та видалення елементів. Вони організовані в ієрархічний спосіб, де кожен вузол представляє пару ключ-значення, що нагадує вигляд дерева.*

Найвищий вузол у дереві називається кореневим вузлом, і він служить початковою точкою для всіх операцій. Кожен вузол містить у собі вказівники на дочірні вузли. Дочірні вузли далі розгалужуються на власні піддерева, створюючи рекурсивну структуру.

Операції вставки та видалення в деревах пошуку зберігають властивість упорядкування дерева. Коли вставляється нова пара ключ-значення, алгоритм порівнює ключ із ключами вузлів, щоб визначити відповідне місце для вставки. Процес триває рекурсивно, доки не буде знайдено порожнє місце, куди можна вставити новий вузол. Видалення вузла з дерева пошуку передбачає пошук вузла для видалення, а потім реорганізацію дерева для збереження властивості впорядкування. Основна ідея полягає в тому, щоб замінити видалений вузол його наступником або попередником, щоб зберегти впорядкованість.

Ефективність пошуку залежить від балансу дерева. Коли дерево ідеально збалансоване, висота мінімізується, що призводить до ефективних операцій пошуку, вставки та видалення з часовою складністю $O(\log n)$. Однак у найгіршому випадку, коли дерево сильно перекошене, операції можуть зайняти $O(n)$ часу.

Для вирішення проблеми балансу та підвищення ефективності операцій у різних сценаріях розроблено різні типи дерев пошуку, наприклад дерева AVL, червоно-чорні дерева та B-дерева. Ці варіанти вводять додаткові пра-

вила та обмеження для підтримки балансу та оптимізації продуктивності на основі конкретних вимог. Наведемо декілька прикладів дерев пошуку та зберігання даних.

Бінарне дерево пошуку.

Визначення 1.1.2 *Бінарне дерево пошуку — це впорядкована структура даних, де кожен вузол містить значення елементу та два можливих вузли-нащадки: лівий та правий. Значення в лівому піддереві менше за значення у батьківському вузлі, а значення в правому піддереві — більше.*

Двійкові дерева пошуку будуються, щоб задовольнити таку властивість: двійкове дерево пошуку — це бінарне дерево з мітками, яке або порожнє, або має позначений кореневий вузол і (i) усі мітки в лівому піддереві менші за кореневу мітку; (ii) усі мітки в правому піддереві більші за кореневу мітку; (iii) ліве та праве піддерева також є двійковими деревами пошуку [1].

Кожен вузол описується таким набором значень:

$$\text{вузол} = \{\text{власне значення, лівий нащадок, правий нащадок}\}$$

Коли дерево побудовано з n елементів, його висота h_n (довжина найдовшого шляху від кореня до будь-якого листка) задовольняє нерівність

$$\lfloor \log_2 n \rfloor \leq h_n \leq n - 1 \tag{1.1}$$

Границі цієї нерівності досяжні; тобто є деякі вхідні послідовності, які дають нижню межу, і є інші послідовності, які дають верхню межу, а також багато послідовностей, які дадуть дерева з висотою між двома межами. Алгоритм побудови бінарного дерева використовується з надією, що результуюче дерево буде досить добре збалансованим і дані можна швидко отримати. Іншими словами, що результуюча найбільша висота ближча до нижньої межі нерівності, ніж до верхньої, і майбутня обробка дерева, що включає пошук одного ключа або вставлення іншого ключа, виконуватиметься в $O(\log n)$, а не в $O(n)$. Завжди є надія, що продуктивність дерева в

найгіршому випадку знаходиться в розумних межах. Зазвичай, коли дерева будуються з рівноймовірних випадкових перестановок їхніх вхідних даних, що є розподілом ймовірностей, найбільш відповідним для використання дерев пошуку як структур даних, більшість дерев є майже збалансованими, а випадки висоти $O(n)$ трапляються рідко.

Дерево квадрантів.

Визначення 1.1.3 *Дерево квадрантів — вид дерева для пошуку та зберігання даних, де кожен внутрішній вузол має по чотири нащадки.*

Нехай дана послідовність незалежних випадкових векторів (U_1, V_1), $(U_2, V_2), \ldots, (U_n, V_n)$, де кожен із векторів має рівномірний розподіл на $[0,1]^2$. Можна побудувати на цих векторах дерево квадрантів так: перша точка з набору даних відповідає кореню дерева квадрантів. Вона розбиває одиничний квадрат на чотири області відповідно до чотирьох географічних напрямків NW, NE, SE і SW, яким ставляться у відповідність цифри $1, 2, 3, 4$. Далі розглядається позиція другої точки відносно першої: в залежності від номера квадранту, вона записується в певний номер нащадка першої точки. Розділ точок повторюється рекурсивно для всього набору даних, і таким чином генерується дерево квадрантів із n внутрішніх вузлів [3].

Визначимо X_n як загальну кількість порівнянь, необхідних для побудови дерева квадрантів за набором векторів $(U_1, V_1), (U_2, V_2), \ldots, (U_n, V_n)$ розміру n. Також X_n — загальна довжина шляху даного дерева квадрантів. Тоді маємо

$$X_1 = 0, \quad X_n \stackrel{\mathrm{d}}{=} n - 1 + X^{(1)}_{I_n^{(1)}} + X^{(2)}_{I_n^{(2)}} + X^{(3)}_{I_n^{(3)}} + X^{(4)}_{I_n^{(4)}}, \quad n \geq 2 \tag{1.2}$$

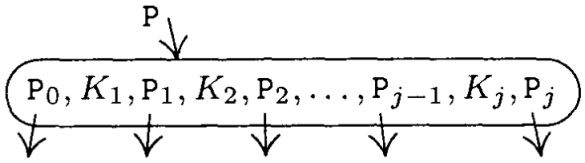
де вектор $\left(I_n^{(1)}, I_n^{(2)}, I_n^{(3)}, I_n^{(4)}\right)$ має змішаний поліноміальний розподіл з параметрами $((n-1, (U(1-V), (1-U)(1-V), (1-U)V, UV))$, де (U, V) має рівномірний розподіл у $[0,1]^2$. Вектор $\left(I_n^{(1)}, I_n^{(2)}, I_n^{(3)}, I_n^{(4)}\right)$ представляє кількість точок, які потрапляють в області $1, 2, 3$ і 4 відносно першої точки (U, V).

B-дерево. B-дерево — структура даних, яка дозволяє ефективне збереження інформації на магнітних дисках та інших пристроях з прямим доступом. Такі дерева зазвичай є збалансованими.

Визначення 1.1.4 *B-дерево порядку m — це дерево, яке задовольняє такі властивості [5]:*

- *Кожен вузол має щонайбільше m дітей.*
- *Кожен вузол, окрім кореня та листя, має принаймні $m/2$ дітей.*
- *Корінь має принаймні 2 дітей (якщо це не лист).*
- *Усі листки з'являються на одному рівні й не несуть інформації.*
- *Нелистовий вузол із k дочірніми елементами містить $k-1$ ключів.*

Вузол, у якому міститься j ключів та $j+1$ вказівників, може бути визначений ось так:



Мал. 1. Стандартний вигляд вузла B-дерева

Червоно-чорне дерево.

Визначення 1.1.5 *Червоно-чорне дерево — це бінарне дерево пошуку, яке має такі властивості:*

- *Кожен вузол або червоний, або чорний.*
- *Кожен листок чорний.*
- *Якщо вузол червоний, то обидва його дочірні елементи чорні.*
- *Кожен простий шлях від вузла до листка-нащадка містить однакову кількість чорних вузлів.*

Коли дерево змінюється, то воно «перефарбовується», щоб відновити властивості забарвлення. Властивості розроблено таким чином, щоб це перевпорядкування та перефарбування можна було виконувати ефективно. Червоно-чорні дерева лишаються збалансованими у динамічному середовищі, і їх можна повторно збалансувати за час $O(\log n)$ [5].

1.2 Метричні дерева

Численні проблеми вимагають ефективної ідентифікації елементів із скінченного набору точок, які знаходяться в певній близькості від заданої точки запиту. Ефективним вважається алгоритм, який дозволяє уникати перевірки кожної точки в наборі. У нагоді стає така структура даних як метричне дерево.

В метричних деревах для пошуку даних використовується поняття відстані між об'єктами, тобто метрика. Це може бути евклідова відстань, манхеттенська відстань або будь-яка інша відстань, визначена як метрика.

Метричні дерева організовують дані у вигляді ієрархічної структури з вузлами та піддеревами. Піддерево з найвищим вузлом "корінь" та вузлами-дітьми "лівий нащадок", "правий нащадок" в записі визначається так:

$$\text{піддерево} = \{\text{корінь}, \text{лівий нащадок}, \text{правий нащадок}\}$$

Метричні дерева будуються рекурсивно. Розподіл простору на елементи дерева відбувається від більших областей на менші за певною умовою (яка залежить від типу дерева). Таким чином, на кожному кроці відбувається розподілення кожної наступної точки набору даних, і після закінченню алгоритму кожен елемент належить певному вузлу дерева. Розподіл відбувається за таким принципом, що близькі об'єкти розташовані ближче один до одного у дереві, щоб полегшити пошук та інші операції.

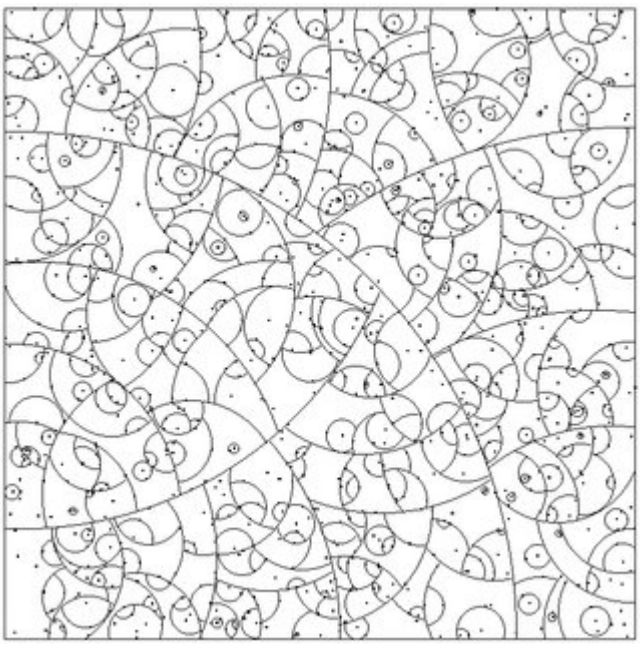
Розглянемо деякі приклади метричних дерев.

k-d-дерева. У k-d-деревах кожен вузол має два нащадки. 1-d-дерево — це звичайне двійкове дерево пошуку; 2-d дерево схоже, але вузли на парних рівнях порівнюють x-координати, а вузли на непарних рівнях порівнюють y-координати під час розгалуження. Загалом, k-d-дерево має вузли з k ко-

ординатами, а розгалуження на кожному рівні базується лише на одній із координат; наприклад, можна проводити розгалуження за числом координат $(k \bmod l) + 1$ на рівні l.

Можна користуватися правилом встановлення зв'язків, заснованим на порядковому номері запису або місці в пам'яті, щоб забезпечити, що жодні два записи не збігаються в будь-якому координатному положенні.
Випадково згенеровані $k-d$-дерева мають точно таку саму середню довжину шляху та розподіл форми, як і звичайні двійкові дерева пошуку [4].

vp-дерева. Як і k-d-дерево, кожен вузол vp-дерева розділяє простір. Але замість того, щоб використовувати значення координат, вузол vp-дерева використовує відстань від вибраної точки огляду. Близькі точки складають лівий, тобто внутрішній підпростір, тоді як правий (зовнішній) підпростір складається з далеких точок. «Близькість» визначається за певною метрикою, і для кожної точки обирається радіус, який визначає лівий підпростір. Рекурсивно формується бінарне дерево, де кожен із вузлів ідентифікує точку огляду (vantage point), а для своїх дітей (лівого та правого нащадка) вузол містить межі пов'язаного з ними підпростору. Щоб побудувати ці структури, метричний простір розкладають за допомогою великих розрізів із формою, що залежить від конкретної обраної метрики. Рандомізовані алгоритми для побудови vp-дерева виконуються за $O(n \cdot \log(n))$ часу [2].



Мал. 2. Приклад декомпозиції vp-дерева

Приклади деяких метрик, які можуть використовуватися для визначення відстані в метричних деревах:

1. Евклідова відстань: найпопулярніша метрика, яка використовується для вимірювання відстані між двома точками в n-вимірному евклідовому просторі. Вона обчислюється за такою формулою:

$$dist_{\text{евкл.}}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{1.3}$$

   де x і y — точки у n-вимірному просторі, а x_i і y_i — їх координати у вимірі i.

2. Манхеттенська відстань: вимірює суму абсолютних різниць між координатами двох точок. Формула для обчислення манхеттенської відстані між точками x і y має вигляд:

$$dist_{\text{манх.}}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{1.4}$$

3. Відстань Чебишова (максимум-метрика): визначається як найбільша різниця між координатами двох точок у просторі. Ця метрика корисна для випадків, коли важлива лише найбільша відстань між точками за якимось виміром. Формула для чебишовської відстані між точками x і y виглядає так:

$$dist_{\text{чеб.}}(x, y) = \max_{i=1}^{n} |x_i - y_i| \tag{1.5}$$

4. Косинусна відстань: використовується для вимірювання схожості між векторами шляхом обчислення косинуса кута між ними. Ця метрика часто використовується в задачах класифікації текстів та рекомендаційних системах. Формула для косинусної відстані між векторами a і b має такий вигляд:

$$dist_{\text{кос.}}(a, b) = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{1.6}$$

5. Метрика Мінковського: загальна форма різних метрик, таких як евклідова, манхеттенська та чебишовська. Вона визначається параметром p, який визначає ступінь метрики. Її формула для точок x і y має такий вигляд:

$$dist_{\text{мінк.}}(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{1.7}$$

Наведені вище метрики використовуються для вимірювання відстані між об'єктами у просторі даних. Зазвичай у практичних випадках метрика обирається в залежності від конкретної задачі та типу доступних даних.

Одним з основних застосувань метричних дерев є пошук найближчих сусідів до певної точки у просторі даних. Пошук найближчих сусідів у метричних деревах може бути здійснений за допомогою алгоритму найближчого сусіда (nearest neighbor) або алгоритму k найближчих сусідів (k nearest neighbors). Обидва алгоритми базуються на ідеї порівняння відстаней між об'єктами у просторі даних.

Алгоритм найближчого сусіда (nearest neighbor):

1. Почати з кореня: обчислити відстань між шуканим елементом і коренем дерева.

2. Рекурсивно переходити до лівого або правого піддерева залежно від того, яке піддерево містить елемент, що ближче до шуканого.

3. Повторювати попередній крок алгоритму до тих пір, поки в результаті не буде отримано листок.

4. Повернути значення листка.

Алгоритм k найближчих сусідів (k nearest neighbors) аналогічний попередньому, але на останньому кроці повертається не один листок, а ще k значень попередніх вузлів.

Ці алгоритми ефективні на метричних деревах, оскільки через їх структуру менша кількість порівнянь, які необхідно виконати. Вони особливо корисні при обробці великих обсягів даних, де прямий перебір всіх об'єктів стає непрактичним.

Щоб підсумувати, метричні дерева можуть використовуватися для таких завдань:

1. Ефективний пошук даних. Однією з основних переваг метричних дерев є те, що вони дозволяють швидко знаходити найближчі точки в просторі до даної за заданою метрикою.

2. Висока масштабованість. Метричні дерева зазвичай добре масштабуються до великих обсягів даних, навіть коли мова йде про мільйони або мільярди елементів.

3. Кластеризація. Метричні дерева можуть бути використані для кластеризації даних, тобто групування елементів за певною ознакою.

4. Використання в індексах баз даних. Метричні дерева можуть бути імплементовані в індексах баз даних для прискорення пошуку за певною метрикою.

5. Можливість імплементування з довільною метрикою, що корисно в тих задачах, де метрика може бути задана через особливі властивості даних.

В цілому, метричні дерева допомагають раціоналізувати та прискорити операції обробки даних у вимірних просторах. Вони забезпечують швидкий доступ до інформації, полегшують аналіз даних та сприяють розвитку різних галузей, де ефективна робота з великими обсягами даних є критично важливою.

2 Структура випадкових vp-дерев

Дано скінченний набір X n-вимірних точок із заданою метрикою $dist(x_i, x_j)$, $x_i, x_j \in X$. Нехай ці точки незалежні та рівномірно випадково розподілені, і отримане дерево буде випадковим. *vp*-дерево $\mathcal{T}$ будується на X рекурсивно наступним чином [2]:

1. Якщо $|X| = 0$, то створюється порожнє дерево $\mathcal{T} = \emptyset$.

2. В іншому випадку обирається довільний елемент x_1 із X, до нього обирається порогове значення τ. Цей елемент x_1 та τ зберігаються в кореневому вузлі дерева.

3. Вставка елементу x_n у дерево на n-тому кроці проводиться рекурсивно. Нехай є вузол T, якщо $dist(x_T, x_n) \leq \tau$, то процедура продовжується відносно лівого нащадка T. Якщо ж нерівність не виконується, то процедура продовжується з правим нащадком. Цей процес відбувається рекурсивно, доки нащадок відповідного вузла не буде пустим, тоді туди записуються x_n та відповідне порогове значення.

Порогове значення класично змінюється експоненційним чином. Для його визначення, наприклад, можна використовувати середнє відстаней від кожної точки до вантажної точки [6]. Інший підхід — використовувати таке правило зміни:

$$\tau_T = \tau^h, \tau \in (0, 1) \tag{2.1}$$

де h — це значення, на 1 більше за відстань від кореня до розглядуваного вузла. Також можна замінити h довжиною крайнього лівого шляху в дереві (шлях від кореня до листка, де переходи відбуваються тільки до лівих нащадків).

Евклідова метрика. Процес побудови *vp*-дерева за допомогою евклідової метрики $dist_{\text{евкл.}}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ можна назвати «кульовою декомпозицією» [6]: навколо кожної вершини утворюється поділ простору на кулю радіусом x_T навколо x_n і на доповнення до неї. Таким чином, крайній лівий шлях являє собою послідовність вкладених куль та їх перетинів.

Максимум-метрика. У максимум-метриці $dist_{\text{чеб.}}(x, y) = \max_{i=1}^{n} |x_i - y_i|$ vp-дерева будуються аналогічно, але замість куль, відбувається розділ простору через побудову гіперкубів, а у результаті їх перетинів утворюються гіперпрямокутники.

Метрика косинусів. *vp*-дерево можна будувати за допомогою косинусної метрики $dist_{\text{кос.}}(a, b) = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}}$, тоді замість представлення точок у вигляді координат у геометричному просторі точки буде представлено у вигляді нормалізованих векторів. Кутова відстань між двома векторами обчислюється за допомогою скалярного добутку, і на основі цієї кутової відстані й будується дерево: елементи розподіляються в групи в залежності від косинусної схожості.

vp-дерева можна адаптувати до багатьох метрик, і відрізнятиметься структура тільки в залежності від того, як виглядає відповідна метрична куля в заданому просторі та із заданою функцією відстані.

3 Математичний аналіз випадкових vp-дерев

3.1 Простори опуклих множин та ймовірнісні міри

Нехай E — векторний простір і X — підмножина E. Кажуть, що X є опуклою множиною, якщо для будь-яких двох точок x і y з X виконується $[x, y] \subset X$.

Визначимо також дві властивості опуклих множин, які випливають із визначення.

1. Нехай $(X_i)_{i \in I}$ — довільна сім'я опуклих множин у E. Тоді їх перетин $\cap_{i \in I} X_i$ також є опуклою множиною.

2. Нехай $(X_i)_{i \in I}$ — сімейство опуклих множин в E так, що для будь-яких двох індексів $i \in I$ і $j \in I$ існує індекс $k \in I$, що задовольняє співвідношення
$$X_i \cup X_j \subset X_k \tag{3.1}$$
Тоді об'єднання $\cup_{i \in I} X_i$ також є опуклою множиною.

Нехай $(\Omega, \mathcal{F}, \mathbb{P})$ — ймовірнісний простір і $\mathcal{X} = \{X(\omega)\}$ — множина всіх $\mathcal{F}$-вимірних відображень $X : \Omega \to \mathbb{R}$. Набір випадкових величин $\mathcal{X}$ і основна ймовірнісна міра $\mathbb{P}$ індукують простір граничних розподілів
$\mathcal{P}^1 := \{P_X(\cdot) = \mathbb{P}\{X \in \cdot\} : X \in \mathcal{X}\}$, простір двовимірних розподілів
$\mathcal{P}^2 := \{P_{XY}(\cdot) = \mathbb{P}\{(X, Y) \in \cdot\} : X, Y \in \mathcal{X}\}$ і так далі [3] [7].

Визначення 3.1.1 *Нехай $\mathcal{D}$ — довільна підмножина $\mathcal{X}$ із такою властивістю: якщо $X \in \mathcal{D}$, то $Y \in \mathcal{D}$ для кожного Y так, що $\mathbb{P}\{X = Y\} = 1$. Така множина $\mathcal{D}$ називається* допустимою.

Метрика, визначена на деякій допустимій множині $\mathcal{D} \subset \mathcal{X}$, є відображенням $\mu : \mathcal{D} \times \mathcal{D} \to [0, \infty)$ з такими властивостями: для всіх $X, Y, Z \in \mathcal{D}$ ми маємо

(Z) $\mu(X,Y)=0$, якщо $\mathbb{P}\{X=Y\}=1$;

(S) $\mu(X,Y)=\mu(Y,X)$;

(T) $\mu(X,Y)\mu(X,Z)+\mu(Z,Y)$.

Псевдометрика, визначена на деякій допустимій множині $\mathcal{D}$, є відображенням $\mu:\mathcal{D}\times\mathcal{D}\to[0,\infty)$, для якого виконуються наведені властивості $(S),(T)$ і також виконується $(Z_P)\,\mu(X,Y)=0$, якщо $\mathbb{P}\{X=Y\}=1$ [3].

Для загальної дисперсії маємо формулу

$$\sigma(X,Y)=\frac{1}{2}\int_{\mathbb{R}}\left|\mathrm{d}\left(F_X(t)-F_Y(t)\right)\right|=\frac{1}{2}\int_{\mathbb{R}}\left|F_X(\mathrm{dt})-F_Y(\mathrm{dt})\right|, \tag{3.2}$$

де $|\mu|$ позначає загальну дисперсію міри зі знаком μ, тобто

$$|\mu|(E)=\sup_{\pi}\sum_{A\in\pi}|\mu(A)| \tag{3.3}$$

Супремум тут береться за всіма розбиттями π вимірної множини E на зліченну кількість непересічних вимірних підмножин.

Нехай $\mathcal{D}$ — допустима підмножина $\mathcal{X}$ і нехай

$$\mathcal{P}\left(\mathcal{D}^2\right)=\left\{\mathbb{P}\{(X,Y)\in\cdot\}:(X,Y)\in\mathcal{D}^2\right\} \tag{3.4}$$

множина всіх спільних розподілів векторів $(X,Y)\in\mathcal{D}^2$.

Визначення 3.1.2 *Відображення* $\mu:\mathcal{P}\left(\mathcal{D}^2\right)\to[0,\infty)$ *називається* ймовірнісною метрикою *на* $\mathcal{D}$, *якщо відображення*

$$\mathcal{D}^2\ni(X,Y)\mapsto\mu\left(P_{XY}\right) \tag{3.5}$$

є псевдометрикою на $\mathcal{D}$.

Якщо μ є ймовірнісною метрикою, то відображення 3.5 також позначатиметься μ. Тоді позначення $\mu(X,Y)$ інтерпретується як $\mu\left(P_{XY}\right)$.

Нехай $\mathcal{D}$ — допустима множина; її можна розбити на непересічні підмножини $\mathcal{D}_X$, так що $\mathcal{D}_X$ складаються з усіх випадкових величин із однаковим розподілом P_X.

Аналогічно, множина $\mathcal{D}^2 = \mathcal{D} \times \mathcal{D}$ може бути розбита на непересічні підмножини $\mathcal{D}^2_{XY} = \mathcal{D}_X \times \mathcal{D}_Y$, що складається з усіх пар випадкових величин (X, Y) таких, що X має розподіл P_X, а Y має розподіл P_Y.

Визначення 3.1.3 *Імовірнісна метрика μ, визначена на допустимій множині $\mathcal{D}$, називається* простою, *якщо відображення 3.5 приймає постійне значення на множинах $\mathcal{D}^2_{XY}$, що утворюють непересічний розбиття $\mathcal{D}^2$. Імовірнісна метрика, яка не є простою, називається* складеною ймовірнісною метрикою *[3].*

Проста ймовірнісна метрика μ не може бути метрикою в сенсі наведеного визначення, оскільки властивість (Z) не виконується. Однак цю властивість можна замінити такою:

(Z') $\mu(X, Y) = 0, P_X = P_Y$

Якщо пояснити ншими словами, властивість (Z') означає, що проста метрика μ приймає нульове значення лише на множині $\mathcal{D}^2_{XX}$. Аналогічно, властивість (Z_P) можна послабити до (Z'_P) $\mu(X, Y) = 0$, якщо $P_X = P_Y$.

3.2 Послідовності випадкових множин, повязані з vp-деревами

Розглянемо послідовність множин, які утворюються в крайньому лівому шляху обходу vp-дерев. Нехай дано метричний простір $([-1, 1]^d, dist)$. Пов'яжемо з кожним вузлом $T_h (h \geq 1)$ таку множину $I_h \subset \mathcal{X}$, яка складається з усіх таких точок x, які б додали до лівого піддерева T_h при побудові дерева. Ці послідовності — вкладені, тобто виконується $I_h \subset I_{h-1}$.

Точка x записується в ліве піддерево T_h, якщо $\operatorname{dist}(x, x_{T_h}) \leq \tau^h$. Це дає

$$I_h = I_{h-1} \cap B_{\tau^h}(x_{T_h}), \quad h \geq 1 \tag{3.6}$$

де $B_{\tau^h}(x_{T_h})$ — метрична куля радіуса τ^h і центром x_{T_h}:

$$B_{\tau^h}(x_{T_h}) = \left\{ x \in \mathcal{X} : \operatorname{dist}(x, x_{T_h}) \leq \tau^h \right\} \tag{3.7}$$

Корінь T_1 дерева завжди є частиною крайнього лівого шляху, тому ми можемо природним чином поставити $I_0 := \mathcal{X} = [-1, 1]^d$, який також є кубом. Вигляд цих множин змінюється в залежності від використовуваної метрики. Якщо використовувати евклідову метрику, то множини будуть перетинами гіперкуль, а якщо максимум-метрику — d-вимірними гіперпрямокутниками.

Спробуємо перетворити послідовність I_h. Визначимо іншу послідовність:

$$J_h = (J_{h-1} + unif(J_{h-1})) \cap B_{\tau^h}(0), \tag{3.8}$$

де $unif(J_{h-1})$ означає точку, яку обрали за рівномірним розподілом всередині множини J_{h-1}, а $B_{\tau^h}(0)$ позначає метричну кулю радіусу τ^h з центром у точці 0.

Поділимо члени цієї послідовності на радіус τ^h і отримуємо такий вираз:

$$\tau^{-1} J_h = \tau^{-1}(\tau^{-h+1} J_{h-1} - \tau^{-h+1} unif(J_{h-1})) \cap B_1(0) \tag{3.9}$$

Позначимо $\tau^{-h} J_h$ як X_h, і тоді остаточний вигляд виразу маємо такий:

$$X_h = \tau^{-1}(X_{h-1} - unif(X_{h-1})) \cap B_1(0) \tag{3.10}$$

Перший елемент визначається як одинична куля: $X_0 = B_1(0)$.

Оскільки вигляд кожної наступної множини X_h цієї послідовності (тобто кожен наступний стан послідовності) залежить тільки від попередньої — X_{h-1}, то ця послідовність є ланцюгом Маркова за визначенням [8].

3.3 Ергодичність (Харріс-рекурентність)

Теорема 3.3.1 *Якщо зафіксувати $\alpha \in S$ і покласти*

$$R_\alpha := \inf \{h \geq 1 : X_h = B_1(0) \text{ враховуючи, що } X_0 = \alpha\}. \tag{3.11}$$

Тоді $\mathbb{E} R_\alpha < \infty$.

Згідно з [2], дана теорема виконується для max-метрики на $[-1, 1]^d$, а також є припущення, що вона виконується і для довільної метрики.

Іншими словами, вищезазначена теорема каже, що ланцюг $(X_h)_{h\geq 0}$ потрапляє у стан $B_1(0)$ нескінченно часто. І можна припустити, що час, за який ланцюг повертається у цей стан, є скінченним і має скінченне середнє. Таким чином, можна розділити весь процес зміни X_h на незалежні періоди між потраплянням у стан $B_1(0)$.

Для формулювання основної теореми необхідно зробити декілька визначень (з [8]).

Визначення 3.3.1 *Ланцюг Маркова $(X_h)_{h\geq 0}$ у просторі станів* S *є ланцюгом Харріса, якщо існують дві множини $A, B \subset S$, позитивна функція $q(x,y) \geq \varepsilon > 0$ для $x \in A, y \in B$ і ймовірнісна міра ρ, зосереджена на B, така що виконуються такі дві умови: 1. $\mathbb{P}\{\inf\{h \geq 0 : X_h \in A\} < \infty\} > 0$ для всіх можливих початкових станів $X_0 \in S$; 2. якщо $x \in A$ і $C \subset B$, то $\mathbb{P}\{X_{h+1} \in C \mid X_h = x\} \geq \int_C q(x,y)\rho(dy)$.*

Визначення 3.3.2 *Ланцюг Харріса називається рекурентним, якщо існує стан α такий, що $\mathbb{P}\{\inf\{h \geq 1 : X_h = \alpha\} < \infty\} = 1$, коли $X_0 = \alpha$*

Визначення 3.3.3 *Рекурентний ланцюг Харріса є аперіодичним, якщо найбільший спільний дільник множини $\{h \geq 1 : \mathbb{P}\{X_h = \alpha \mid X_0 = \alpha\} > 0$ дорівнює 1.*

Твердження $(X_h)_{h\geq 0}$ — аперіодичний рекурентний ланцюг Харріса.

За допомогою цього твердження (доведення в [2]) можна сформулювати наступну важливу теорему про властивості $(X_h)_{h\geq 0}$.

Теорема 3.3.2 *Ланцюг Маркова $(X_h)_{h\geq 0}$ має стаціонарний розподіл I_∞ і*

$$\lim_{n\to\infty} |\mathbb{P}\{X_h = \cdot \mid X_0 = B_1(0) - I_\infty(\cdot)\|_{TV} = 0, \tag{3.12}$$

де $\|\cdot\|_{TV}$ позначає загальну відстань дисперсії.

Доведення З твердження вище маємо, що (X_h) є аперіодичним рекурентним ланцюгом Харріса. Згідно з [8] цей ланцюг має стаціонарну міру і збігається до стаціонарного розподілу в загальній дисперсії відстані, якщо $R_{B_1(0)}$ є майже напевно скінченним (виконання чого забезпечується 3.3.1).

4 Моделювання

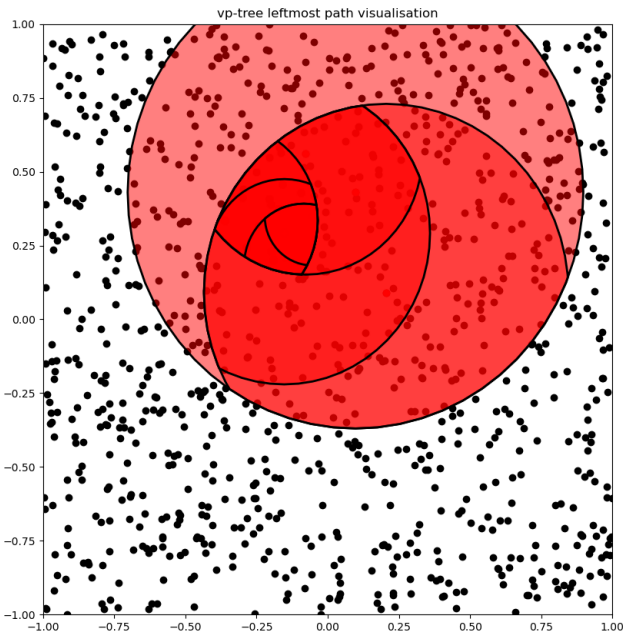
Метою є побудувати структуру vp-дерева, зобразити його крайній лівий шлях та пов'язану з ним послідовність випадкових множин. Приведемо декілька візуалізацій для наглядності, а потім за допомогою моделювання перевіримо, чи виконується те, що процес X_h нескінченно часто повертається в початковий стан $B_0(1)$ і час його повернення в цей початковий стан є скінченним 3.3.2. Моделювання проводиться для евклідової метрики; випадок максимум-метрики детально досліджено в [2].

Експеримент 1.

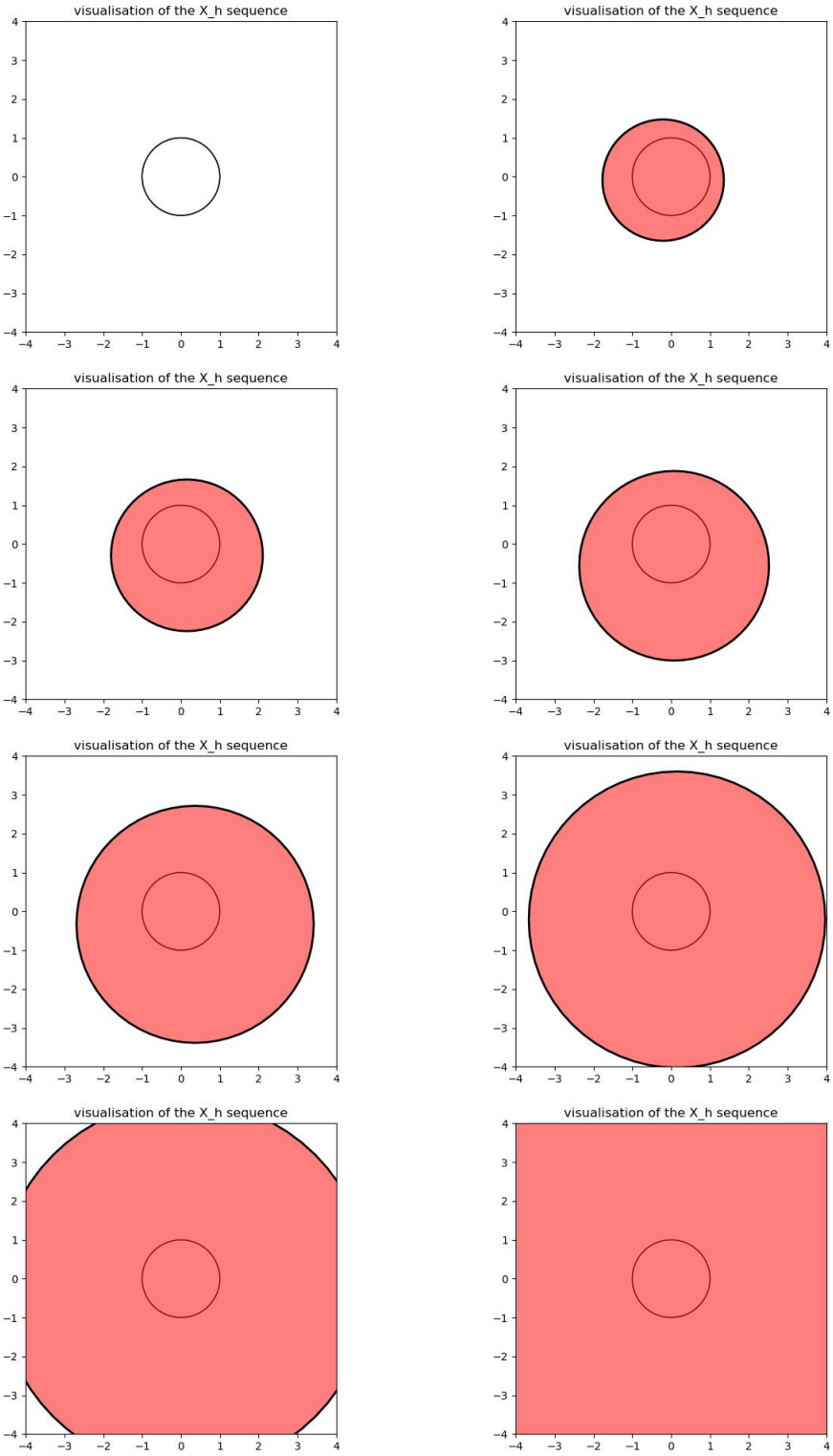
Кількість точок: 1000, τ=0.8

Висота дерева: 19

Довжина крайнього лівого шляху для даного дерева: 8



Мал. 3. Крайній лівий шлях для отриманого дерева.



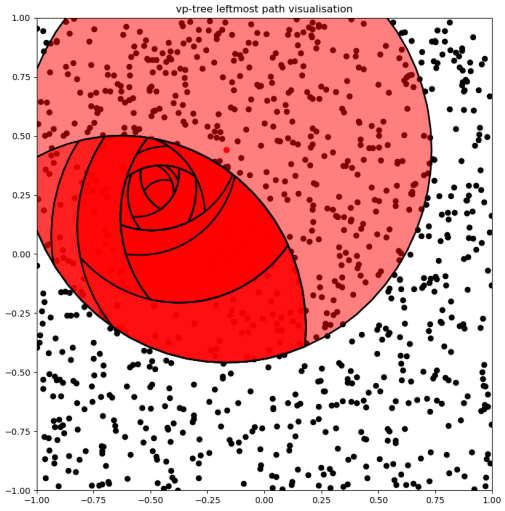
Мал. 4. Зміна послідовності X_h: після розтягування та паралельного переносу, але до перетину з одиничним колом. (Для наглядності, бо сама послідовність X_t має вигляд одиничних кіл.) Можна побачити, що кожна наступна множина в даному випадку містить одиничне коло, а отже перетин із ним і дасть саме те коло.

Експеримент 2.

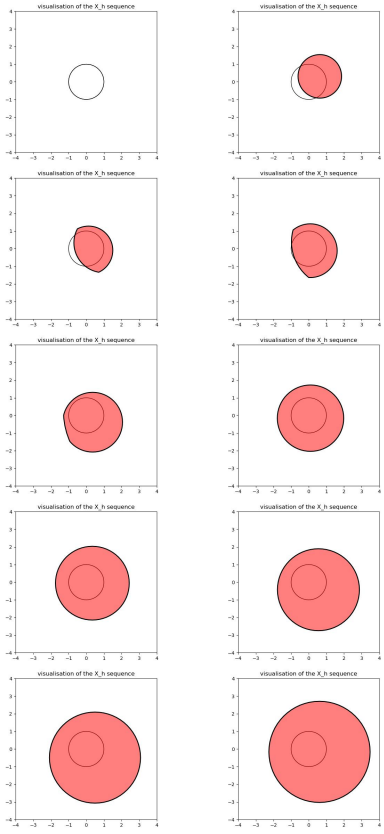
Кількість точок: 1000, τ=0.9

Висота дерева: 28

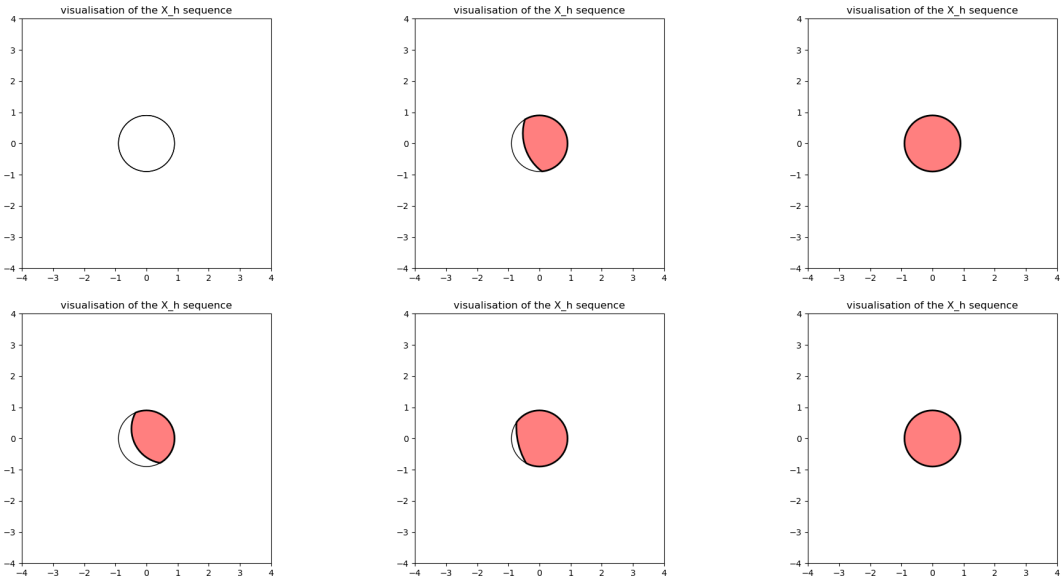
Довжина крайнього лівого шляху для даного дерева: 21



Мал. 5. Крайній лівий шлях для отриманого дерева.



Мал. 6. Зміна послідовності X_h до перетину з одиничним колом.



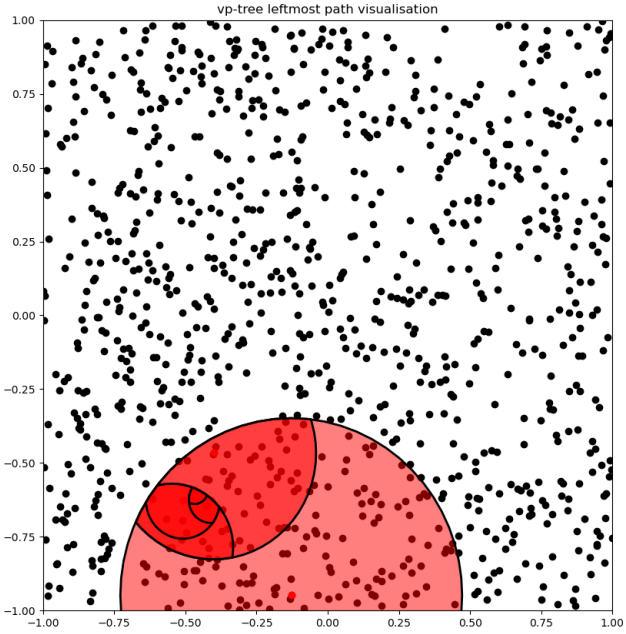
Мал. 7. Вигляд перших елементів послідовності X_h.

Експеримент 3.

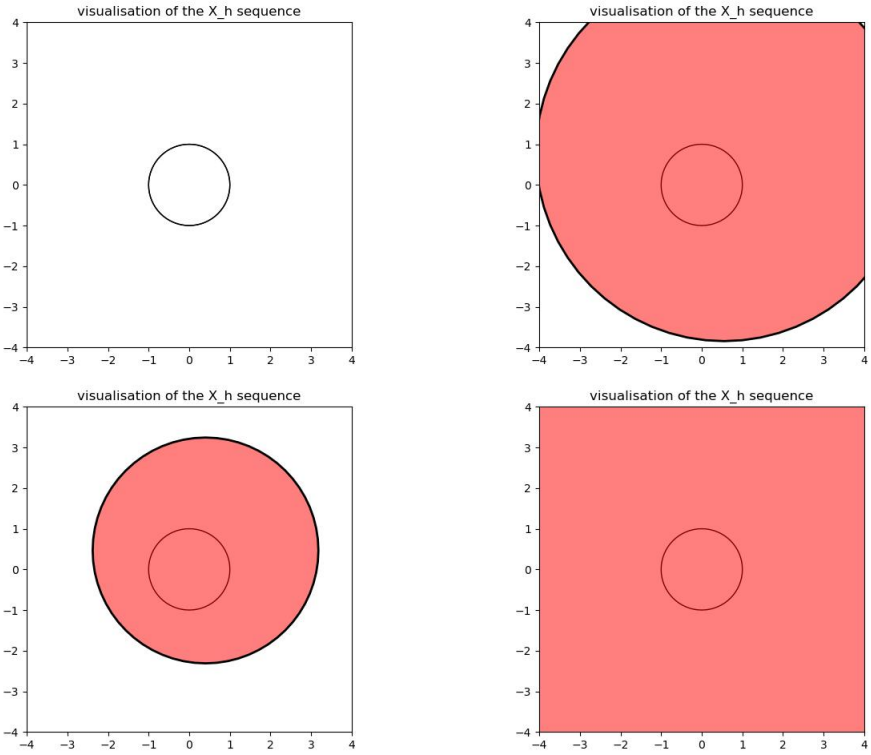
Кількість точок: 1000, τ=0.6

Висота дерева: 18

Довжина крайнього лівого шляху для даного дерева: 6



Мал. 8. Крайній лівий шлях для отриманого дерева.



Мал. 9. Зміна послідовності X_h до перетину з одиничним колом.

Час повернення до початкового стану в залежності від τ та кількості елементів у наборі даних.

tau	0.4	0.5	0.6	0.7	0.8	0.9
n = 1000	0	0	0	0	0	0.17647
n = 2000	0	0	0	0	0	0.09524
n = 5000	0	0	0	0	0.33333	0.13043
n = 10000	0	0	0	0.1	0.16667	0.15385
n = 20000	0	0	0	0	0	0.09375

5 Висновки

Кваліфікаційна робота на здобуття ступеня бакалавра присвячена vp-деревам та їх властивостям. Результати роботи такі:

1. Розглянуто різні структури даних для організації даних, підходи до їх побудови та використання.

2. Проаналізовані властивості vp-дерев, а також пов'язаних із ними випадкових множин та ланцюгів Маркова.

3. Побудовано модель vp-дерева, виконано серію експериментів із моделювання vp-дерев та пов'язаних із ними множин.

Дана робота може бути розвинута далі в напрямку подальшого емпіричного, а згодом і строгого дослідження властивостей vp-дерев, таких як залежність їх збалансованості від обраного радіусу.

Також можна покращувати саме програмний продукт, а саме створити зручний інтерфейс для задання параметрів та метрик.

Бібліоґрафія

[1] Hosam M. Mahmoud, Evolution of random search trees, 1992 — с. 57-59, 178

[2] V. Bohun, Probabilistic analysis of vantage point trees, 2021 — с. 2-7

[3] О. Маринич, Probabilistic analysis of algorithms, 2021 — с. 79-84, 110, 113-114

[4] D. Knuth, The Art of Computer Programming (vol. 3) — с. 426-480, 565

[5] R. Sedgewick, K. Wayne, Algorithms, 2010. Princeton — algs4.cs.princeton.edu/30searching

[6] J. Uhlmann, Metric trees, 1991

[7] V. V. Buldyga, Geometric Aspects of Probability Theory and Mathematical Statistics, 1992 — с. 3-5

[8] Durrett, R.: Probability Theory and Examples, Fourth Edition. Cambridge University Press, New York, 2010. — с. 269-327

Додатки

Код роботи

```
import numpy as np
import shapely.geometry as sg
from matplotlib import pyplot as plt
from matplotlib.patches import Circle
import descartes
from shapely import affinity
from statistics import mean

class Node:
    def __init__(self, x, tau):
        self.left = None
        self.right = None
        self.data = (x, tau)

class Tree:
    def __init__(self, tau):
        self.counter = 0
        self.tau = tau

    def createNode(self, x, tau):
        return Node(x, tau)

    def insert(self, node, x):
        if node is None:
            return self.createNode(x, self.tau)
        if euclidean(x, node.data[0]) <= node.data[1]:
            self.tau = node.data[1]*TAU
            node.left = self.insert(node.left, x)
        elif euclidean(x, node.data[0]) > node.data[1]:
            self.tau = node.data[1]
            node.right = self.insert(node.right, x)
        return node

    def height(self, root):
        if root is None:
            return 0
```

```
        leftHeight = self.height(root.left)
        rightHeight = self.height(root.right)
        return max(leftHeight, rightHeight) + 1

    def fill(self, root, arr):
        for a in arr:
            self.insert(root, a)

    def traverse(self, child, parent, position):
        # parent is a previous circle
        # child_patch is added at each step to the drawing
        # child is a node: ((x,i), tau)
        # at each step: child_circle is generated, then it is intersected with parent
        # then a patch is created, and this patch is added to the drawing
        # then it recursively calls itself with arguments as follows:
        # next node (child.left), child_circle
        if child is not None:

            child_circle = sg.Point(child.data[0][0], child.data[0][1]).buffer(child.data[1])
            if position == "left":
                child_circle = child_circle.intersection(parent)

            child_patch = gen_patch(child_circle)
            axes.add_patch(child_patch)

            self.traverse(child.left, child_circle, "left")

    def inorderTraversal(self, root):
        res = []
        if root:
            res = self.inorderTraversal(root.left)
            res.append(root.data)
            res = res + self.inorderTraversal(root.right)
        return res

    def traverseLeft(self, node, res):
        if node is not None:
            res.append(node.data)
            self.traverseLeft(node.left, res)


def euclidean(p1, p2):
  return np.sqrt(np.sum(np.power(p2 - p1, 2)))

def gen_patch(x):
```

```
  return descartes.PolygonPatch(x, fill=True, fc=(1,0,0,0.5), ec=(0,0,0,1)
    , lw=2)

def set_graphics(unit):
    plt.xlim(-4, 4)
    plt.ylim(-4, 4)
    plt.title('visualisation of the X_h sequence')
    plt.rcParams["figure.figsize"] = [10, 10]
    unit = sg.Point(0,0).buffer(1)
    unit2 = descartes.PolygonPatch(unit, fill=False, color='black')
    axes.add_patch(unit2)
    axes.set_aspect(1)

TAU = 0.9
xy_min = [-1, -1]
xy_max = [1, 1]
n = 20000 # n = int(input("enter the number of points: "))
# np.random.seed(0)
data = np.random.uniform(low=xy_min, high=xy_max, size=(n,2))

figure, axes = plt.subplots()
plt.figure(1)

# build vp-tree
tree = Tree(TAU)
root = tree.createNode(data[0], TAU)
tree.fill(root, data[1:])

# get nodes of the leftmost path
res = []
tree.traverseLeft(root, res)
x_i = [x[0][0] for x in res] # x coordinates of leftmost path
y_i = [x[0][1] for x in res] # y coordinates of leftmost path
rad = [x[1] for x in res]    # tau
# print(res)

unit = sg.Point(0,0).buffer(1)
unit2 = descartes.PolygonPatch(unit, fill=False, color='black')
set_graphics(unit2)
axes.add_patch(unit2)

plt.tight_layout()
plt.savefig('/home/sunday/plots/file' + '0' + '.png', dpi=100)
axes.clear()
set_graphics(unit2)
n_circles = len(res) # the number of circles to draw
height = tree.height(root)
print("Висота дерева: ", height)
```

```
print("Довжина крайнього лівого шляху для даного дерева: ", n_circles)
circles = [unit]

# calculate steps before chain reaches point zero

zero = []
counter = 0

# draw the rest of the circles
for i in range(1, n_circles):
    center = (x_i[i], y_i[i])

  # modify X_(h-1) to get X_h
    circle = circles[i-1]
    circle = affinity.translate(circle, xoff=-center[0], yoff=-center[1])
    circle = affinity.scale(circle, xfact=1/rad[i], yfact=1/rad[i])

    # actually plot X_h
    circle2 = gen_patch(circle) # draw a patch after translation and
   scaling
    axes.add_patch(circle2)
    if circle.contains(unit):
        zero.append(counter)
        counter = 0
    else:
        counter += 1
    circle = circle.intersection(unit) # intersect this scaled patch with
   a unit circle: get a segment of the unit circle
    # axes.add_patch(gen_patch(circle))
    #circle3 = gen_patch(circle)
    #axes.add_patch(circle3)
    circles.append(circle)
    plt.tight_layout()
    num = str(i)
    if i < 10:
        num = "0" + num
    plt.savefig('/home/sunday/plots/file' + num + '.png', dpi=100)
    axes.clear()
    set_graphics(unit2)

plt.show()

print(mean(zero))

# actually plotting points
figure, axes = plt.subplots()
plt.xlim(-1, 1)
plt.ylim(-1, 1)
```

```
175 plt.scatter(data[:, 0], data[:, 1], color='black')
176 plt.scatter(x_i, y_i, color='red') # nodes of the leftmost path
177 tree.traverse(root, None, None)
178 plt.title('vp-tree leftmost path visualisation')
179 plt.show()
```

Лістинг 5.1: Код для генерації vp-дерева