МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н.Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему Використання графових нейронних мереж
для побудови укладання графа

Виконала: студентка 4 курсу, групи
МФ-41 спеціальності 122
«Комп'ютерні науки», освітньо-
професійної програми
«Інформатика» Лінник О.С.

Керівник: к.ф.-м.н. Полякова Л.Ю.

Харків – 2024

ЗМІСТ

ВСТУП

В сучасному світі, де величезний потік інформації вимагає структурування та аналізу, питання укладки графів набуває особливої важливості. Графи, як математичні структури, можна виявити в усіх аспектах нашого життя, від соціальних мереж і транспортних систем до біологічних взаємодій та інформаційних технологій.

Велика частина складних взаємозв'язків та систем може бути ефективно відображена та проаналізована за допомогою графів. Здатність якісно зобразити граф може зробити значний внесок у розуміння структури та властивостей різноманітних мереж. Зі зростанням обсягу даних, підвищується потреба в нових алгоритмах укладки графів, або ж прискоренні вже існуючих за допомогою сучасних методів, таких як, наприклад, використання нейронних мереж.

Машинне навчання набуло величезної популярності за останні роки, адже його методи визначають нові стандарти у вирішенні завдань складної обробки та аналізу інформації. Застосування цих методів у алгоритмах укладки графів надає унікальні можливості для автоматизації та оптимізації процесу аналізу великих обсягів даних.

Вдосконалення методів укладання графів, застосовуючи графові нейронні мережі є **метою цієї роботи**. Актуальність цього дослідження обумовлена потребою в розробці ефективних і адаптивних методів укладки графів, які здатні впоратися зі зростаючим обсягом даних та їх складною структурою.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Базові поняття

Задача укладки графа в математиці та комп'ютерних науках полягає у визначенні оптимального розташування вершин і ребер графа на площині з метою задоволення певних критеріїв. Оптимальність у даному контексті може означати мінімізацію перетинів ребр, збереження пропорцій відстаней між вершинами, естетичний вигляд графічного представлення та інші параметри відповідно до конкретних вимог і контексту застосування.

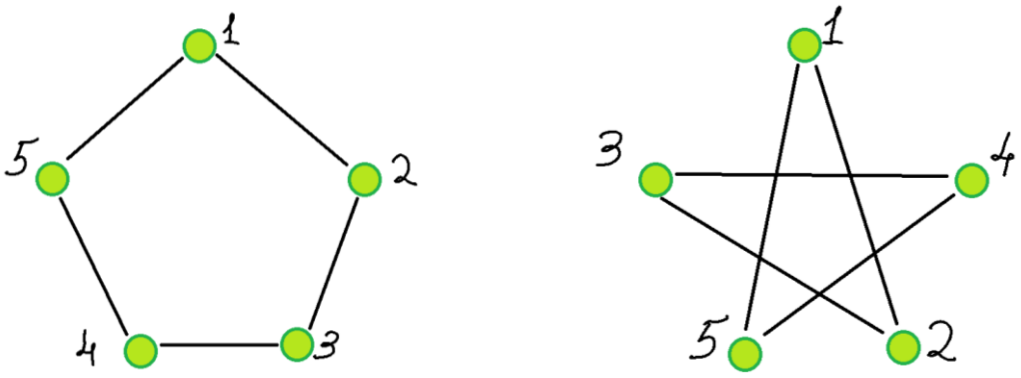
Проблема інформативного зображення графів виникає у багатьох областях, включаючи комп'ютерну науку, телекомунікації, біологію, соціологію та інші. Наприклад, в інформаційних технологіях, укладка графів може бути потрібна для представлення мережі зв'язків між комп'ютерами або веб-сторінками. У біології граф може відображати взаємодії між різними видами організмів або органічних сполук.

Наведемо визначення основних термінів, які нам будуть потрібні. Основні поняття теорії графів наведено за [1].

Граф $G = (V, E)$ складається з множини вершин $V \neq \emptyset$, а також з множини ребер E. Кожне ребро є парою вершин.

Укладка графа на площині – це відображення φ, яке співставляє кожній вершині v точку $\varphi(v)$ на площині, а ребру uv – відрізок $\varphi(u)\varphi(v)$, що сполучає відповідні точки. Аналогічно можна визначити укладку графа у тривимірному просторі. Образи вершин та ребер під дією φ також називають вершинами та ребрами (укладки).

Зауважимо, що один і той самий граф може мати різні укладки. Наприклад, на рисунку 1.а зображено граф правильного п’ятикутника, а на рисунку 1.б цей же п’ятикутник укладено так, що виходить зірочка.



а) *б)*

Рис. 1 Різні укладки одного графу

Зважений граф – це граф із додатковою ваговою функцією на ребрах, яка співставляє кожному ребру деяке число (вагу).

Матриця суміжності графу *G з n вершинами* – це матриця $A = ||a||$ розміру $(n \times n)$, де $a_{ij} = a_{ji} = 1$, якщо між вершинами *i* та *j* є ребро, та $a_{ij} = a_{ji} = 0$, якщо ребра немає. У випадку, коли елемент a_{ij} дорівнює вазі ребра *ij* у зваженому графі (і 0, якщо ребра немає), кажуть про ***вагову матрицю*** графа.

Шлях – послідовність вершин, у якій кожна наступна вершина з'єднана з попередньою ребром.

Зв'язний граф – граф, в якому між будь-якою парою вершин є шлях.

Основні поняття щодо машинного навчання та нейронних мереж наведено за [2].

Нейрон – основна будівельна одиниця нейронних мереж, яка моделює нейрон в мозку. Вона приймає вхідні сигнали, зважує їх і використовує активаційну функцію для генерування вихідного сигналу.

Нейромережа - це математичний модель, яка імітує роботу людського мозку. Вона складається з взаємопов'язаних штучних нейронів, які обробляють інформацію. Існує багато типів нейромереж, включаючи звичайні штучні нейронні мережі (ANN), згорткові нейромережі (CNN), рекурентні нейромережі (RNN) та графові нейромережі (GNN). Всі вони

використовуються в різноманітних областях, таких як комп'ютерне зорове сприйняття, мовний аналіз, рекомендаційні системи, медична діагностика, фінанси та багато інших. Навчання проходить на основі великої кількості даних, використовуючи методи залежно від типу задачі.

Існує два основних ***підходи до навчання*** – кероване і некероване. У керованому навчанні модель навчається за допомогою звичайних пар вхідних і вихідних даних, де передбачення моделі порівнюється з правильними відповідями, і коригується помилка, щоб покращити результат. У некерованому навчанні модель самостійно вивчає складності даних без явного використання вихідних міток. Цей підхід дозволяє моделі знаходити приховані структури в даних та робити узагальнені висновки для випадків, де не існує однозначно правильної відповіді.

Функція активації – функція, яка використовується в кожному штучному нейроні для обчислення вихідного значення на основі зважених вхідних сигналів.

Навчання – процес, під час якого модель нейронної мережі вдосконалюється шляхом зміни внутрішніх параметрів за допомогою тренувальних даних.

Функція втрат – це функція, яка вимірює різницю між прогнозованими значеннями моделі та фактичними значеннями в тренувальних даних.

Алгоритм оптимізації – алгоритм, який використовується для налаштування параметрів моделі з метою мінімізації функції втрат.

2.2. Основні алгоритми

Почнемо з огляду історії зображення графів, який було описано у статті [3]. Перші малюнки повноцінних графів були зроблені, ще в 14-му столітті. Це були зображення фамільних дерев різних величних родів. Але в ті часи це не сприймалося як граф в математичному сенсі. Хоча саме поняття «граф» було введено лише в 1878 році англійським математиком

Джеймсом Джозефом Сильвестром, перші зображення математичних графів з'явились ще в 16-му столітті. Приклади можна подивитись у [3].

Одним з перших математиків, що розглянув задачу укладки графів був Вільям Татт. В його статті 1963 року «Як намалювати граф» [4] він описує ідеї силового вкладання, які потім були розвинуті багатьма іншими вченими.

Розглянемо декілька алгоритмів укладки зв'язних графів. Зауважимо, що задачу оптимальної укладки різних компонент зв'язності графа розглядають окремо (див., наприклад, [5]) після того, як знайдено укладку кожної компоненти. У даній роботі ми не будемо приділяти увагу цьому питанню. Кожен з алгоритмів оптимізує розташування вершин графу на площині. Критерієм якості укладки може бути мінімальна кількість перетинів ребр, вдала передача структури графу (наприклад, виділення кластерів), мінімальна площа опуклого багатокутника, в який можна помістити всі вершини укладки, тощо.

2.2.1. Алгоритм Камади-Каваї

Алгоритм Камада-Каваї (див. [6]) ґрунтується на певній фізичній інтерпретації графа. Нехай ми маємо систему V з n вузлів $\{v_1, v_2, \dots, v_n\}$, які поєднані між собою пружинами. На площині поставимо у відповідність кожному вузлу v_i вершину укладки p_i, отримаємо множину $\{p_1, p_2, \dots, p_n\}$. Вершини поєднаємо ребрами відповідно до системи V. Вважатимемо, що кожне ребро буде мати вагу, яка дорівнює його довжині. Фізична інтерпретація ребер як пружин, дозволила Камада та Каваї ввести поняття енергії системи та вимірювати за допомогою нього дисбаланс системи. Основне припущення полягає в тому, що мінімум енергії системи відповідає оптимальній укладці графа.

Енергія системи задається наступним чином:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} \left(|p_i - p_j| - l_{ij} \right)^2. \quad (1)$$

Для розуміння цієї формули нам також знадобиться d_{ij} – довжина мінімального шляху між вузлами i та j. Тоді:

- $k_{ij} = \frac{K}{d_{ij}^2}$ – сила пружності між вузлами i, j, якщо вони з'єднані пружиною, або ж сила відштовхування, якщо ні (K – константа для масштабування зображення системи);
- $L = L_0 / \max\limits_{i<j} d_{ij}$ (L_0 – довжина горизонталі екрану) – оптимальна довжина одного ребра;
- $l_{ij} = L \times d_{ij}$ – «ідеальна» відстань між вузлами на зображенні;
- $|p_i - p_j|$ – поточна відстань між вузлами на зображенні (де p_i та p_j задано координатами).

Тепер перейдемо до алгоритму зображення системи оптимальним чином. Система буде представлена у вигляді графа, який ми будемо вкладати на площину, отже нехай кожна вершина графа p_i має координати $(x_i; y_i), i \in [1..n]$. Тоді формулу (1) можна переписати наступним чином:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} \left(\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right) - l_{ij} \right)^2 = \tag{2}$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} k_{ij} \frac{(x_i - x_j)^2 + (y_i - y_j)^2 - 2l_{ij}\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + l_{ij}^2}{2}$$

Таким чином енергія перетворюється на функцію $E(x_1, \dots, x_n, y_1, \dots, y_n)$, яку треба мінімізувати. Метод Камада та Каваї передбачає пошук локального мінімуму замість глобального, використовуючи метод Ньютона-Рафсона [7]. Необхідною умовою локального мінімуму є наступна рівність:

$$\frac{\partial E}{\partial x_m} = \frac{\partial E}{\partial y_m} = 0, \text{для } 1 \le m \le n.$$

$$\frac{\partial E}{\partial x_m} = \sum_{i \neq m} k_{mi} \left((x_m - x_i) - \frac{l_{mi}(x_m - x_i)}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}} \right) \quad (3)$$

$$\frac{\partial E}{\partial y_m} = \sum_{i \neq m} k_{mi} \left((y_m - y_i) - \frac{l_{mi}(y_m - y_i)}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}} \right) \quad (4)$$

Тобто, щоб знайти локальний мінімум E нам потрібно розв'язати систему 2n нелінійних рівнянь з (3) та (4), які можуть залежати одне від одного. У цьому випадку 2n-значний метод Ньютона-Рафсона не підходить, але якщо рухати на площині за один крок лише одну вершину, а інші зафіксувати, то ми отримаємо функцію двох координат ($E(x_m; y_m)$) і локальний мінімум такої функції можна шукати двозначним методом Ньютона-Рафсона.

Розглянемо $\Delta_m = \sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2}$. Оптимальним буде на кожному кроці рухати вершину, в якої Δ_m найбільше. Позначимо початкову позицію вершини p_m як $(x_m^0; y_m^0)$. На кожній ітерації будемо робити наступне:

$$x_m^{t+1} = x_m^t + \delta x, \qquad y_m^{t+1} = y_m^t + \delta y, \qquad \text{для } t = 1, 2, 3 \ldots \quad (5)$$

Шукати δx та δy будемо з системи:

$$\begin{cases} \frac{\partial^2 E}{\partial x_m^2}(x_m^t; y_m^t)\delta x + \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m^t; y_m^t)\delta y = -\frac{\partial E}{\partial x_m}(x_m^t; y_m^t) \\ \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m^t; y_m^t)\delta x + \frac{\partial^2 E}{\partial y_m^2}(x_m^t; y_m^t)\delta y = -\frac{\partial E}{\partial y_m}(x_m^t; y_m^t) \end{cases} \quad (6)$$

Коефіцієнти цієї системи можна знайти з матриці Якобіана, отриманої з частинних похідних (3) та (4) за x_m та y_m:

$$\frac{\partial^2 E}{\partial x_m^2} = \sum_{i \neq m} k_{mi} \left(1 - \frac{l_{mi}(y_m - y)^2}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}^3} \right) \quad (7)$$

$$\frac{\partial^2 E}{\partial x_m \partial y_m} = \sum_{i \neq m} k_{mi} \left(\frac{l_{mi}(x_m - x_i)(y_m - y_i)}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}^3} \right) \quad (8)$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m} = \sum_{i \neq m} k_{mi} \left(\frac{l_{mi}(x_m - x_i)(y_m - y_i)}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}^3} \right) \quad (9)$$

$$\frac{\partial^2 E}{\partial y_m^2} = \sum_{i \neq m} k_{mi} \left(1 - \frac{l_{mi}(x_m - x)^2}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}^3} \right) \quad (10)$$

Закінчити ітерації треба тоді, коли Δ_m буде достатньо малим.

Отже, алгоритм можна описати наступним чином:

1. Рахуємо d_{ij} для кожної пари вершин в графі. (Наприклад, алгоритмом Флойда [8])
2. Рахуємо l_{ij}, k_{ij} для кожної пари вершин в графі.
3. Ініціалізуємо вершини $p_i, i \in 1..n$. Камада та Каваї пропонують для цього розташування по колу з діаметром L_0.
4. Запускаємо мінімізацію:

while ($\max_i \Delta_i > \varepsilon$){

знаходимо p_m: $\Delta_m = \max_i \Delta_i$;

while ($\Delta_m > \varepsilon$){

рахуємо δx *та* δy *з (6);*

$x_m := x_m + \delta x$;

$y_m := y_m + \delta y$;

}}

2.2.2. Алгоритм Фрухтермана-Рейнгольда

В основі алгоритма Фрухтермана-Рейнгольда (див., наприклад, [9]) також лежить фізична модель графа, у якій вершини розглядають як систему частинок, які відштовхуються і притягуються одна до одної під дією певних сил. Зазвичай використовуються сили пружності, що базуються на законі Гука, щоб притягувати пари кінцевих точок ребер графа одна до одної, тоді як відштовхувальні сили, подібні до сил електрично заряджених частинок за законом Кулона, використовуються для

відштовхування всіх пар вершин. У рівноважних станах для цієї системи сил ребра зазвичай мають однакову довжину (через пружні сили), а вершини, не з'єднані ребром, зазвичай відштовхуються одна від одної (через електричне відштовхування). Сили притягування ребер і відштовхування вершин можуть бути визначені за допомогою функцій, що не базуються на фізичній поведінці пружин і частинок; наприклад, деякі системи на основі сил використовують пружини, притягувальна сила яких є логарифмічною, а не лінійною. Оптимальною укладкою графу вважається та, якій відповідає мінімум результуючої сили. Далі наведено кроки алгоритму:

1. Ініціалізація: На початку кожна вершина графа розміщується на площині випадковим чином. Це створює початковий розподіл вершин.
2. Обчислення відштовхуючих сил: Кожна вершина відштовхує інші вершини. Сила відштовхування між двома вершинами залежить від їх поточного розташування і налаштована так, щоб вершини не перетиналися. Зазвичай використовується закон обернено пропорційного відстані: чим ближче вершини, тим сильніше вони відштовхуються одна від одної.
3. Обчислення сил тяжіння: Кожна вершина також притягується до центру графа (або до інших центральних вершин, якщо це вказано). Це допомагає запобігти розсіюванню вершин. Сила тяжіння зазвичай зменшується зі збільшенням відстані до центру.
4. Переміщення вершин: На основі обчислених відштовхуючих і притягуючих сил вершини переміщуються на невелику відстань в кожній ітерації. Чим сильніше відштовхування або притягування, тим більше переміщення.
5. Ітерації: Кроки 2-4 повторюються кілька разів (зазвичай фіксована кількість або до досягнення критерію зупинки, наприклад, поки сума

всіх сил не стане достатньо малою).

6. Візуалізація: Після завершення ітерацій вершини розміщуються на площині, і графічна візуалізація графа готова.

Алгоритм Фрухтермана-Рейнгольда широко використовується для візуалізації графів у різних галузях, таких як аналіз соціальних мереж, біоінформатика, візуалізація Інтернет-шляхів та інше. Він забезпечує більш рівномірний та естетичний розподіл вершин на площині порівняно з більш простими методами, такими як випадкове розміщення. Ребра укладки, побудованої алгоритмом Фрухтермана-Рейнгольда зазвичай мають приблизно однакову довжину, на відміну від алгоритма Камада-Каваї, який може утворити укладку, довжини ребер якої наближаються до наперед заданих.

2.2.3. Алгоритми Барабаши

У цьому розділі ми розглянемо поєднання ідеї моделювання графа фізичною системою з ідеєю моделювання обчислення укладки за допомогою нейронної мережі.

2.2.3.1. Загальні ідеї

Сучасні алгоритми укладки графів найчастіше поєднують ідеї методів Камада-Каваї та Фрухтермана-Рейнгольда та використовують ту чи іншу фізичну модель графів для побудови так званої сило-спрямованої укладки (Force Directed Layout – FDL). Недоліком цих алгоритмів є висока обчислювальна складність на великих обсягах даних та часто незрозуміла структура у випадку графів з великою кількістю вершин та ребер. Альбертом Барабаши та іншими ([10]) було запропоновано підхід з використанням графових нейронних мереж який прискорює FDL в 10-100 разів та робить візуалізацію більш інформативною.

У своїй модифікації FDL Барабаши використовує графові згорткові мережі (Graph Convolutional Networks – GCN). Цей клас нейромереж призначений для обробки графових даних. У порівнянні зі звичайними

зв'язними штучними нейронними мережами (ANNs), які призначені для обробки послідовних або сіткових даних, GCN можуть ефективно моделювати взаємодію між вершинами (вузлами) фізичної моделі графа. Основна ідея полягає у тому, щоб кожен вузол у графі представити вершиною укладки в просторі, положення якої обчислюється на основі інформації про її сусідів. Під час навчання GCN використовує інформацію про структуру графа, щоб врахувати контекст і зв'язки між вузлами при розподілі ваг нейронів.

2.2.3.2. Методи

Далі ми спочатку опишемо алгоритм FDL в класичній модифікаціїї та в модифікації Барабаши, а потім перейдемо до застосування нейронних мереж для створення попереднього вкладення вершин графа (в алгоритмах NeuLay-2 та NodeMLP), на якому потім застосовується метод FDL Опис наступних методів було взято зі статті [10] Альберта Барабаши та інших авторів.

Спочатку опишемо класичний алгоритм FDL. Нехай є неорієнтований зважений граф з N вершин, заданий матрицею суміжності $A \in \mathbb{R}^{N\times N}$, де $A_{ij} = 1$, якщо є ребро між вершинами i та j та $A_{ij} = 0$, якщо ребра немає. Також визначимо множину $X = \{x_1, x_2, \dots, x_N\}$, елемент x_i якої буде вектором з координатами вершини під номером i у d-вимірному евклідовому просторі. FDL змінює координати вершин, мінімізуючи загальну енергію системи $\mathcal{L}$ (тобто $\mathcal{L}$ є функцією втрат), що складається з двох частин (V_{el} , V_{NN}), які визначають відповідно енергії еластичності та відштовхування.

$$\mathcal{L}(X) = V_{el} + V_{NN}, \qquad V_{el} = \frac{1}{2}\sum_{i,j} A_{ij}\left|x_i - x_j\right|^2 = \frac{1}{2}Tr[X^T L X] \tag{11}$$

$L = D - A$ – лапласіан графа, який задається за допомогою матриці $D_{ij} = \delta_{ij}\sum_k A_{ik}$ – матриці степенів вершин. Для визначення V_{NN} є різні варіанти.

У класичному FDL використовується більш повільний варіант:

$$V_{NN}(X) = a_N \sum_{i,j} r_0 / \|x_i - x_j\| \quad (12)$$

a_N та r_0 – параметри фізичної моделі, які відповідають за жорсткість та розмір пружини.

Такий варіант функції втрат буде рахуватись за $O(N^2)$. Якщо використати при обчисленнях алгоритм оптимізації Барнса – Хата, можна знизити час розрахунків до $O(NlogN)$. Цей алгоритм рекурсивно будує дерево степеня 4, структура якого дозволяє швидше діставатись до необхідних вершин [11].

Автори алгоритму NeuLay-2 вирішили використати більш швидкий, але в деяких випадках менш ефективний спосіб обчислення сили відштовхування:

$$V_{NN}(X) = a_N \sum_{i,j} \exp(|x_i - x_j|^2 / 4r_0^2) \quad (13)$$

a_N та r_0 – параметри фізичної моделі, які відповідають за жорсткість та розмір пружини.

Таке $V_{NN}(X)$ буде обчислюватись за $O(N)$. Автори зазначають, що саме в їх алгоритмі використовувати (12) не обов'язково. Далі в класичному підході до розрахунку укладки за допомогою FDL запускається градієнтний спуск для мінімізації $\mathcal{L}$ за допомогою пересувань вершин і таким чином знаходиться оптимальна укладка графа.

В алгоритмі Барабаши також основною метою є мінімізація $\mathcal{L}$, але для переміщення вершин в просторі використовуються глибокі нейронні мережі. Для цього створюється дві архітектури: NeuLay-2 та NodeMLP.

1) NodeMLP починає з випадкового Z – вкладення вершин в простір $N \times h$ (де $N -$ кількість вершин). Далі алгоритм шукає оптимальну проекцію цього розташування в тривимірний простір $(N \times d, d = 3)$. Обчислення відбувається наступним чином: $X = \sigma(ZW + b)$, де σ –

нелінійна функція, Z, W, b – параметри нейронної мережі, які треба натренувати.

2) NeuLay використовує GNN (Graph Neural Network), яка починає з випадкового Z – вкладення вершин в простір $N \times h$ (де $N -$ кількість вершин) і застосовує GCN (Graph Convolutional Network) шар для обчислення:

$$G_1 = \sigma\left(f(A)ZW^{(1)}\right), f(A) = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}, \tilde{A} = A + I, \tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \quad (14)$$

$\tilde{D}_{ij}$ – матриця степенів, I – одинична матриця. G_1 є новою укладкою вершин у h_1 – вимірному просторі та має розміри $N \times h_1$.

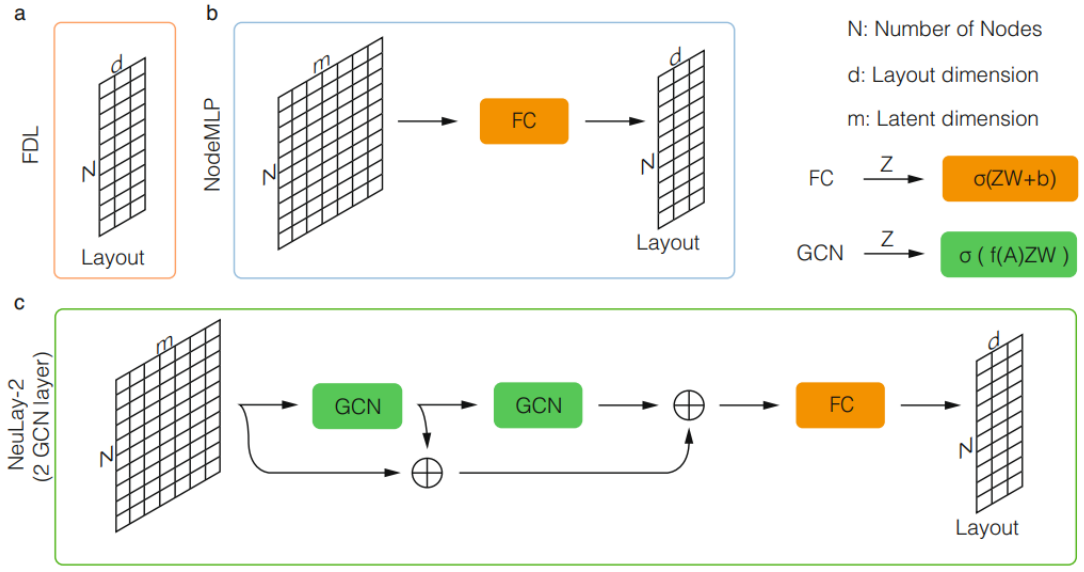


Рис. 2 Схеми алгоритмів FDL, NodeMLP та NeuLay-2 згідно з [10]

На рисунку 2 автори схематично зображують простори, з якими працюють алгоритми FDL, NodeMLP та NeuLay-2, та показують, що застосовується при переході від однієї до іншої розмірності.

NeuLay-2 додатково застосовує ще один шар GCN та отримує укладку $G_2 = \sigma\left(f(A)ZW^{(2)}\right)$ розміру $N \times h_2$ у h_2–вимірному просторі.

Далі автори пропонують об'єднати всі шари наступним чином: $G = (Z|G_1|G_2)$, отримуючи укладку в $(h + h_1 + h_2)$–вимірному просторі.

Нарешті автори застосовують NodeMLP для отриманої $(h + h_1 +$

h_2)–вимірної укладки графа, щоб отримати проекцію G у тривимірний простір – остаточну укладку. Множина параметрів для тренування у NeuLay-2:

$$\theta = \{Z, W^{(1)}, W^{(2)}, W, b\}. \quad (15)$$

Щоб знайти оптимальну укладку, автори дають розташування $X(\theta)$ на вхід алгоритму FDL, й оптимізують укладку, використовуючи функцію втрат $\mathcal{L}$, яка була задана у формулах (11-13).

Замість прямої оптимізації X, автори пропонують оптимізувати θ, користуючись наступною формулою:

$$\frac{d\theta_a}{dt} = -\varepsilon \frac{d\mathcal{L}}{d\theta_a} = -\varepsilon \sum_i \frac{dx_i d\mathcal{L}}{d\theta_a dx_i}. \quad (16)$$

На відміну від звичайного використання нейронних мереж, де параметри тренуються лише один раз, автори тренують θ для кожної укладки.

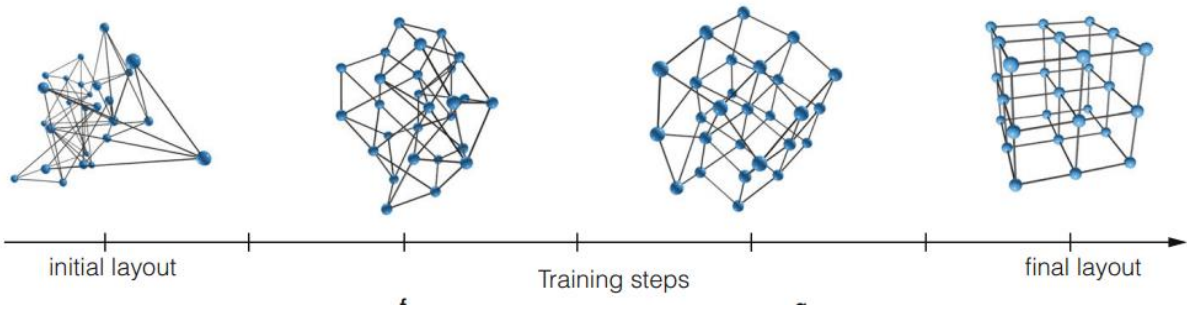


Рис. 3 Дія алгоритму NeuLay-2 згідно з[10]

На рисунку 3 автори показують, що відбувається з укладкою куба протягом дії алгоритму NeuLay-2.

У коді, який автори надали для відкритого доступу в своїй статті, не було додано можливість зображувати зважені графи. Також, у відкритих даних, які використовували автори алгоритму NeuLay-2 для тестування, всі графи незважені, а отже, можна припустити, що автори не розглядали випадок зваженого графу для свого алгоритму.

3. АЛГОРИТМ KAMADANN

У цьому розділі ми запропонуємо алгоритм побудови укладки графу KAMADANN, який поєднує ідею Барабаши щодо моделювання початкової укладки за допомогою графової нейронної мережі з алгоритмом Камада-Каваї.

3.1. Гіпотеза

При вивчені існуючих алгоритмів укладки графів, було висунуто наступну гіпотезу: ідеї прискорення FDL за допомогою машинного навчання можна застосувати і до інших алгоритмів укладки графів, що може прискорити їх. Таким алгоритмом було обрано метод Камада-Каваї, який дуже схожий на FDL. Головною їх відмінністю є визначення функції втрат.

Було зроблено припущення, що для занурення алгоритму Камада-Каваї у NeuLay-2 достатньо змінити визначення функції втрат та внести деякі зміни у побудову моделі графа. Таким чином можна отримати пришвидшену версію алгоритму Камада-Каваї, який також буде мати перевагу над NeuLay-2 – можливість працювати зі зваженими графами.

3.2. Опис алгоритму

Основна відмінність алгоритму KamadaNN від NeuLay-2 – визначення функції втрат. Ми вирішили використати загальну енергію системи, яку визначили Камада та Каваї [6]. Її було наведено у формулі (1):

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} \left(\left| p_i - p_j \right| - l_{ij} \right)^2. \quad (1)$$

У цій формулі

d_{ij} – довжина мінімального шляху між вершинами графа i та j,

$k_{ij} = \frac{K}{d_{ij}^2}$ – сила пружності між вершинами, якщо вони з'єднані ребром (уявною пружиною), або ж сила відштовхування, якщо ні (K – константа для масштабування зображення системи, їй надали значення 1), $L =$

$L_0 / \max_{i<j} d_{ij}$ (де L_0 – довжина горизонталі екрану, їй також встановили значення 1) – оптимальна довжина одного ребра,

$l_{ij} = L \times d_{ij}$ – «ідеальна» відстань між вершинами в укладці,

$|p_i - p_j|$ – поточна відстань між вершинами в укладці (p_i та p_j задано координати, за допомогою яких рахується ця величина).

Також, важливою відмінністю цієї моделі від моделі Барабаши є зваженість графа, отже у ваговій матриці нашої моделі кожен елемент є вагою ребра між відповідними вершинами. Це дозволяє будувати укладки графа, довжини ребер яких наближаються до наперед заданих.

Інших змін в NeuLay-2 не було внесено. Таким чином, алгоритм KamadaNN фактично є злиттям алгоритмів Камада-Каваї та NeuLay-2.

Для кращої візуалізації графів було додано більше методів зображення готової укладки. За допомогою бібліотек plotly [12] та matplotlib [13] було створено наступні методи:

1. Зображення 2d графу у двовимірному просторі.
2. Зображення 3d графу у двовимірному просторі.
3. Динамічне зображення 3d графу у тривимірному просторі.

3.3. Схема алгоритму

Ініціалізація моделі в багатовимірному просторі:

1. створення матриці суміжності
2. випадкове розміщення графу в просторі
3. створення матриці відстаней методом Флойда
4. створення матриці обернених квадратичних відстаней
5. масштабування

↓

Запуск першого GCN з функцією втрат Камада-Каваї

↓

Запуск другого GCN з функцією втрат Камада-Каваї

↓

Злиття трьох отриманих шарів графа в один шар за допомогою методу

NodeMLP

↓

Проекція отриманого розташування з багатовимірного простору у дво/тривимірний

↓

Відображення укладки графа залежно від його розмірності

3.4. Реалізація алгоритму

Реалізація алгоритму KamadaNN була здійснена мовою програмування Python. Список основних бібліотек та модулів, які були використані мною:

- Бібліотека PyTorch [17] зберігає дані у вигляді тензорів та надає інструменти для швидких обчислень на таких структурах даних.
- Модуль torch.nn та бібліотека PyTorch Geometric [18] надають інструменти для роботи зі звичайними та графовими нейронними мережами відповідно.
- Бібліотека SciPy [19] надає інструменти для оптимізації коду.
- Бібліотека NetworkX [15] надає інструменти для роботи з графами.
- Бібліотеки Plotly [12] та Matplotlib [13] надають інструменти для візуалізації даних.

При ініціалізації моделі я додала створення матриці d, обчислення якої відбувається алгоритмом Флойда (його реалізовано в бібліотеці SciPy). Далі на основі цієї матриці обчислюються матриці k та l. Також при кожному обчисленні поточної загальної енергії системи, за допомогою методу cdist бібліотеки PyTorch, створюється матриця distances, яка містить в собі поточні відстані між усіма парами вершин графу.

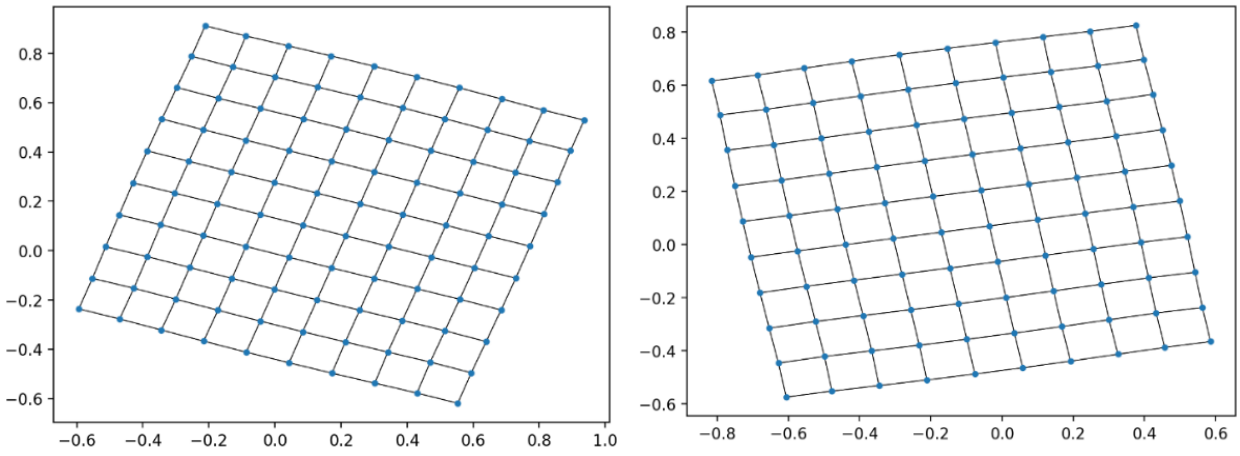
Я розділила метод для візуалізації укладки, який надають автори NeuLay-2, на три окремих методи: 2d укладка, 3d укладка та динамічна 3d укладка (реалізовано за допомогою plotly.graph_objects).

4. ЕКСПЕРИМЕНТ

Алгоритм KamadaNN пройшов тестування на різноманітних графах, з метою дослідження його здатності відображати різні аспекти структури графів. Під час цього тестування вивчалася здатність алгоритму до відтворення симетрії, формування кластерів, збереження ваги зв'язків та інших ключових характеристик структури графа. Результати тестування наведено нижче. Обчислення часу виконання програми проводились за допомогою методу time_it, який було реалізовано авторами алгоритму NeuLay-2. Вказаний час – усереднення трьох запусків програми.

4.1. Симетрія

Для початку розглянемо графи, які мають виражені симетричні риси або чітку структуру.



а) *б)*

Рис. 4 Граф «волейбольна сітка 10 на 10 2d», два запуски

На рисунках 4а та 4б зображено результати двох запусків алгоритму на графі «волейбольна сітка 10 на 10» у 2d режимі. Модель приймає на вхід параметр DIMENSIONS = 2 і робить проекцію не в тривимірний (як було описано вище), а в двовимірний простір. Алгоритм відпрацьовує в середньому за 6 секунд. Як можна побачити, алгоритм в точності передає структуру сітки та зберігає симетрію.

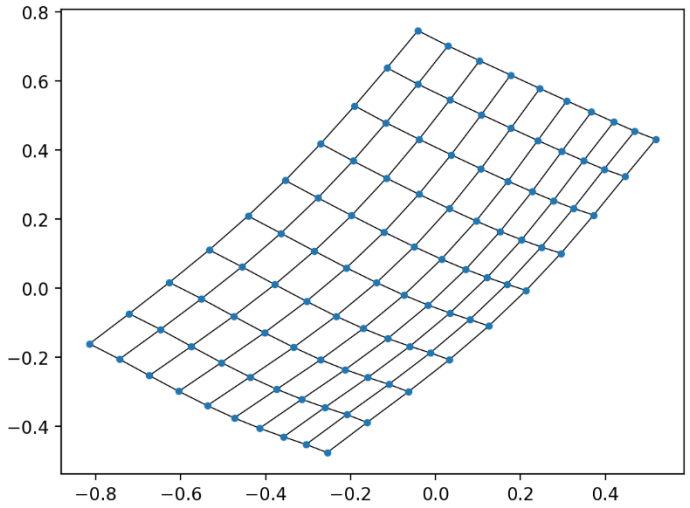


Рис. 5 Граф «волейбольна сітка 10 на 10 3d»

На рисунку 5 зображено результат виконання алгоритму на графі сітці, але в режимі 3d. В такому випадку алгоритм працює у тривимірному просторі, а потім робить випадкову проекцію з 3d у 2d. При цьому, ми отримуємо 2d-структуру у вигнутому вигляді.

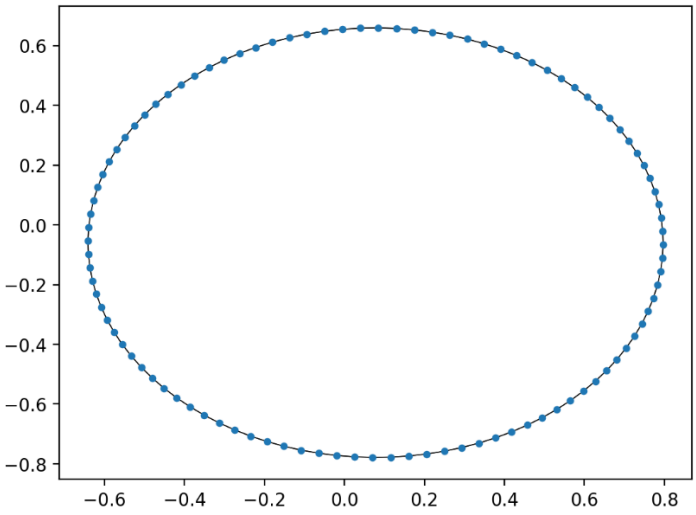
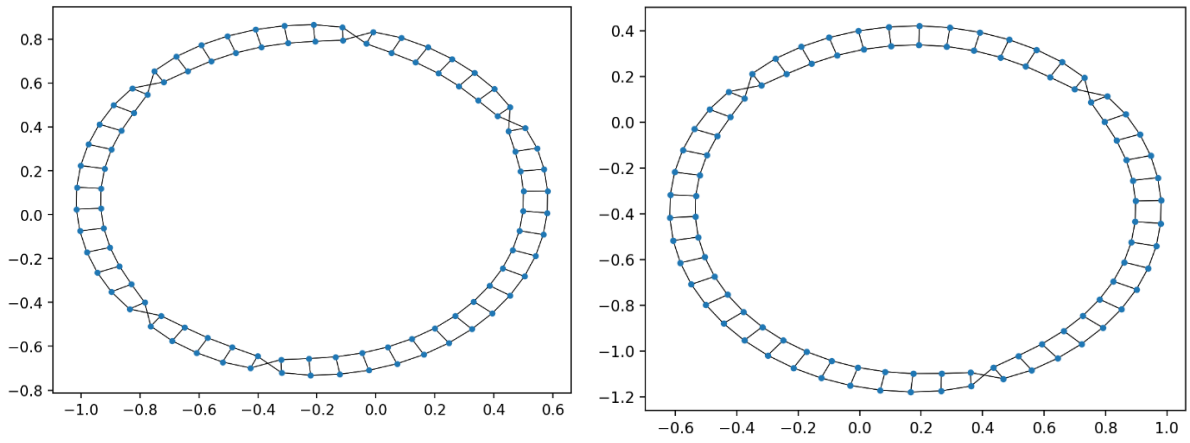


Рис. 6 Граф «правильний 100-кутник»

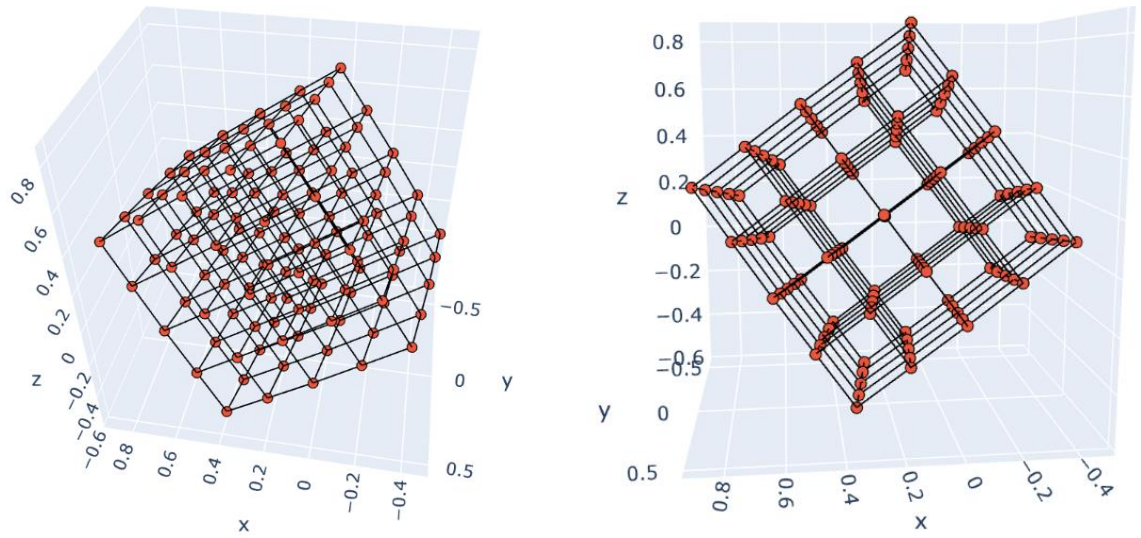
На рисунку 6 відображено результат виконання алгоритму на графі «правильний 100-кутник». Програма відпрацьовує за 6 секунд. Структура передана прекрасно, симетрія збережена.



а) *б)*

Рис. 7 Граф «100-кутне колесо», два запуски

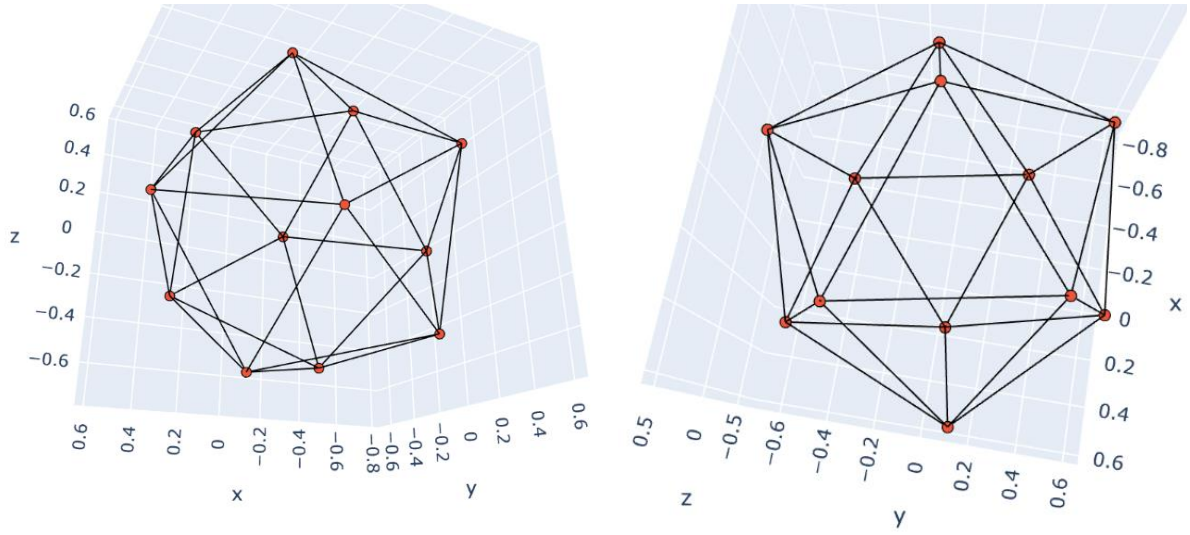
На рисунках 7а та 7б відображено результати двох запусків алгоритму на графі «100-кутне колесо». Це граф правильного 100-кутника, в якому також з’єднані протилежні вершини. Алгоритм працює в середньому 6.5 секунд. Результат відрізняється від задуманої структури через те, що ми намагаємось уникати перетинів ребр, в той час як в запланованій структурі кількість перетинів дуже велика. Центральна симетрія частково втрачена.



а) *б)*

Рис. 8 Граф «Куб 5 на 5 на 5».

На рисунках 8а та 8б відображено результат виконання алгоритму на графі кубу 5 на 5 на 5 в двох проекціях. Програма відпрацювала за 4 секунди. Структура куба передана ідеально.



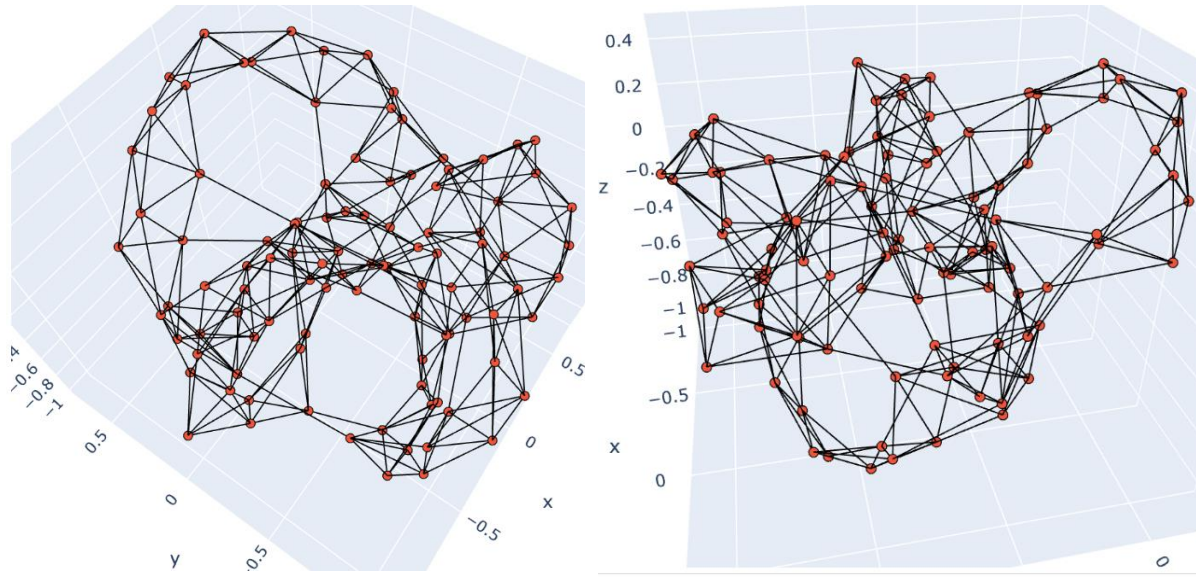
а) *б)*

Рис. 9 Граф «ікосаедр».

На рисунках 9а та 9б відображено результат виконання алгоритму на графі «ікосаедр» в двох проекціях. Програма відпрацювала за 3 секунди. Структура ікосаедру успішно передана.

4.2. Випадкова геометрія

Для моделювання випадкової геометрії було використано граф Воттса-Строгаца [14]. Такий граф має високу кластеризацію.

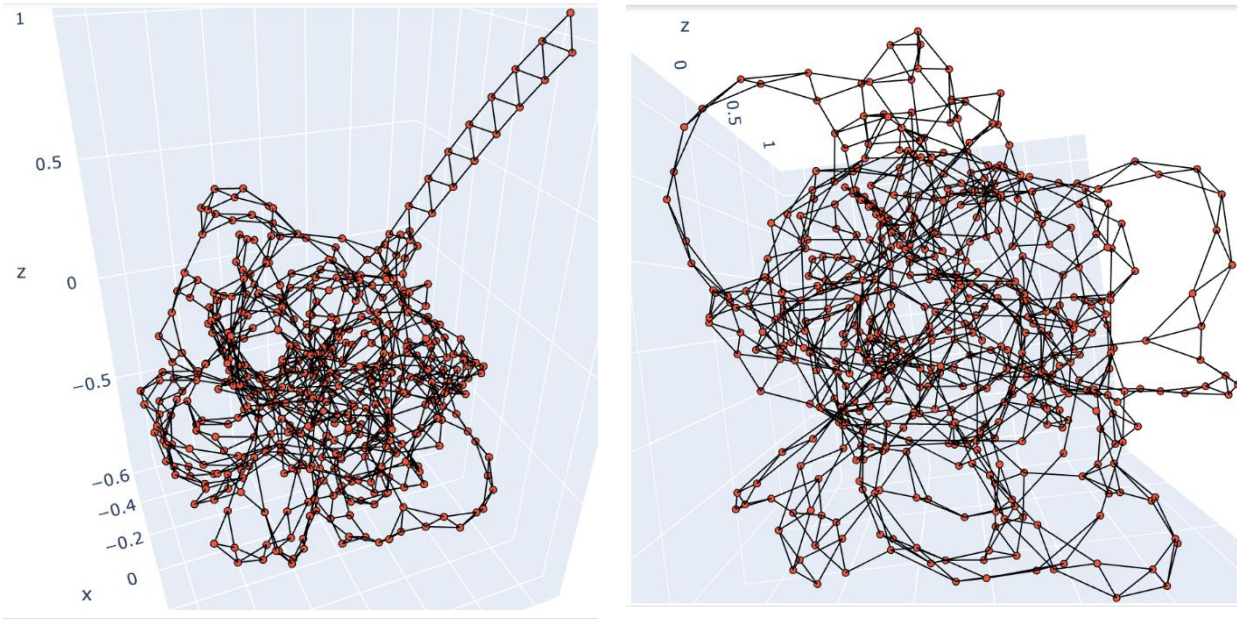


а) *б)*

Рис. 10 Граф «Воттс-Строгац (100, 7, 0.1)»

На рисунках 10а та 10б зображено результат виконання програми для

графа «Воттс-Строгац (100, 7, 0.1)» в двох проекціях. Це граф на 100 вершинах, де кожна вершина спочатку має 7 сусідів і кожен зв’язок може переорієнтуватись з ймовірністю 0,1. Середній час виконання програми 4 секунди.



а) *б)*

Рис. 11 Граф «Воттс-Строгац (500, 4, 0.1)»

На рисунках 11а та 11б зображено результат виконання програми для графу «Воттс-Строгац (500, 4, 0.1)» в двох різних проекціях. Це граф на 500 вершинах, де кожна вершина спочатку має 4 сусіди і кожен зв’язок може переорієнтуватись з ймовірністю 0,1. Середній час виконання програми складає 34,5 секунди.

На тестових зображеннях видно чітке групування об'єктів в різні структури, що свідчить про наявність кластерів. Отже, можна вважати, що алгоритм KamadaNN ефективно відображає цю особливістю.

4.3. Зважені графи

Алгоритм KamadaNN у своїй функції втрат враховує вагу ребр графа, тому були проведені тести саме на графах зі зваженими ребрами, щоб перевірити ефективність алгоритму у відображенні таких структур. Результати тестувань наведено нижче.

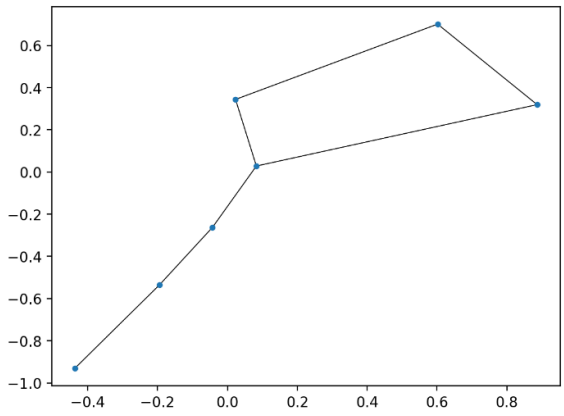


Рис. 12 Граф «Велика ведмедиця»

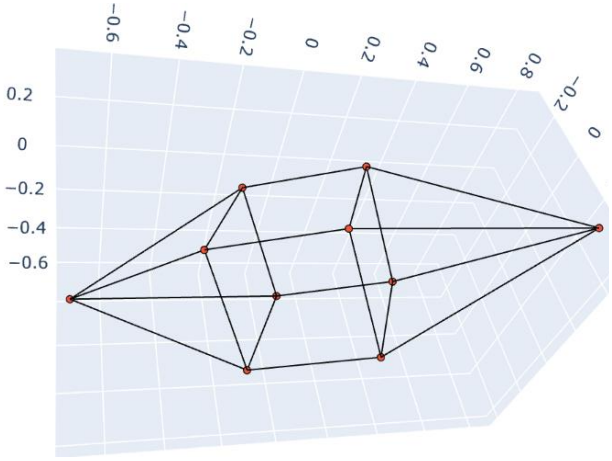
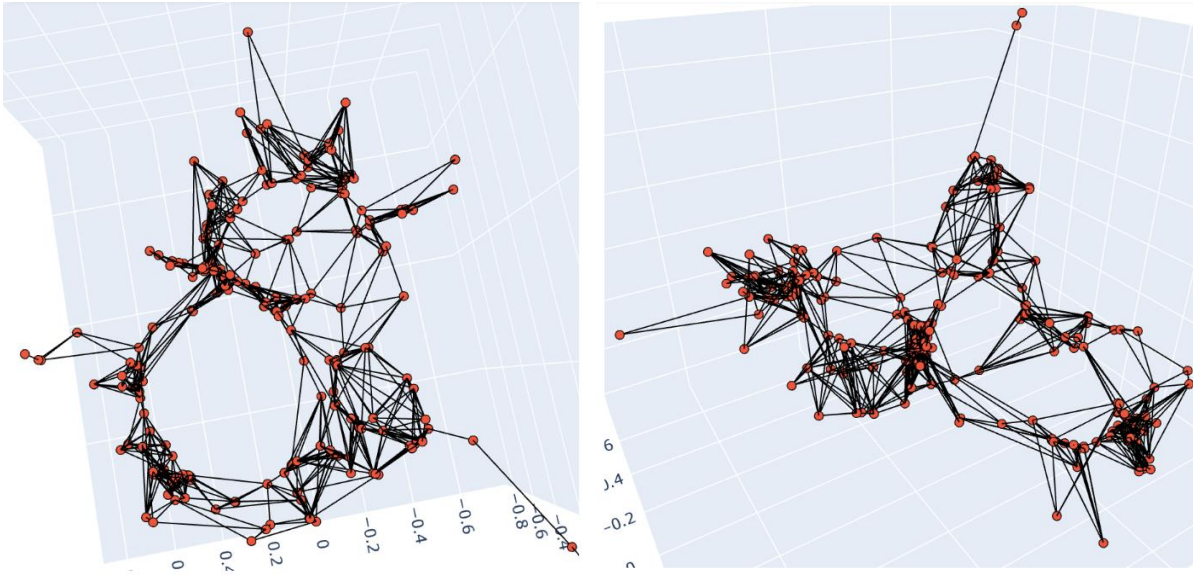


Рис. 13 Граф «Витягнутий кристал»

На рисунку 12 зображено укладку графу «велика ведмедиця». Його було задано матрицею суміжності з вагами ребр, які приблизно відповідають відносним відстаням у сузір'ї великої ведмедиці. Алгоритм побудував сузір'я перегорнутим, але дуже схожим на оригінал.

На рисунку 13 зображено 3d укладку «витягнутого кристалу». Його було задано матрицею суміжності з такими вагами ребр: кубик в середині має всі ребра 1, а ще дві вершини з'єднуються з кубиком ребрами ваги 2.

Обидва зображення гарно передають зваженість.



а) *б)*

Рис. 14 Граф «Випадкова геометрія (200, 0.125)»

На рисунках 14а та 14б зображено граф «випадкова геометрія (200, 0.125)» в двох різних проекціях. Цей граф має 200 вершин, а коефіцієнт 0.125 відповідає за радіус створення ребр. Після створення, кожному ребру було дано випадкову вагу на інтервалі (0, 1). На малюнках гарно видно кластеризацію, а також помітно, що граф зважений. Середній час виконання програми на таких даних складає 24 секунди.

4.4. Інші відомі графи

Далі наведено результати експериментів на різних відомих графах. Для цього було використано графи з python-бібліотеки NetworkX[15], яка є однією з найкращих у роботі з графами і надає величезну кількість інструментів для їх побудови та дослідження.

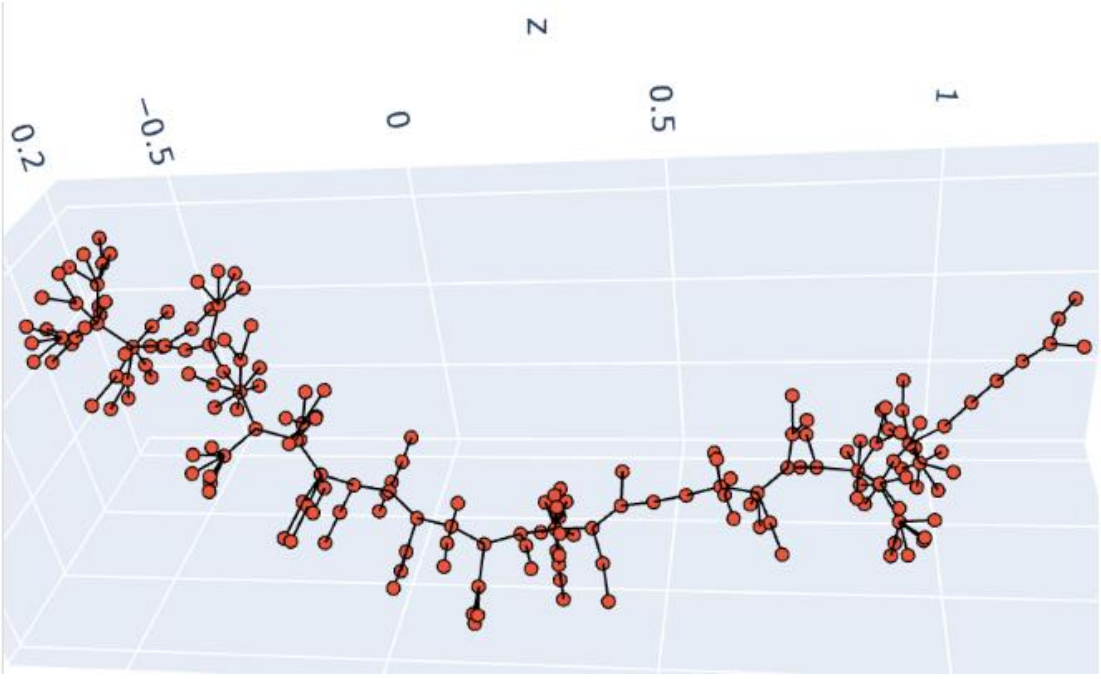


Рис. 15 [Граф «Лобстер (100, 0.6, 0.5)»]

На рисунку 15 можна побачити граф Лобстера [16] зі 100 вершин, з такими коефіцієнтами для створення ребер: 0.6 – ймовірність створення ребра на хребті та 0.5 – за його межами. Середній час роботи програми на таких даних – 12 секунд.

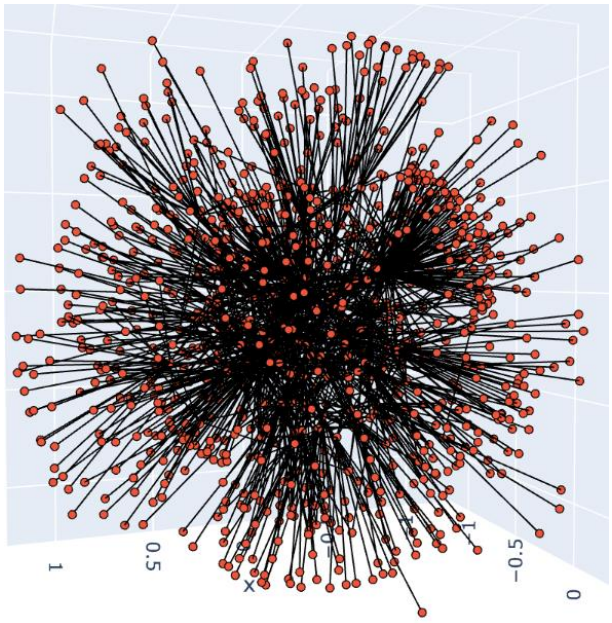


Рис. 16 Граф «Інтернет 1000»

На рисунку 16 відображено модель інтернету з 1000 вершин. Алгоритм працює в середньому за 59 секунд. Структура отриманого результату відповідає очікуваній.

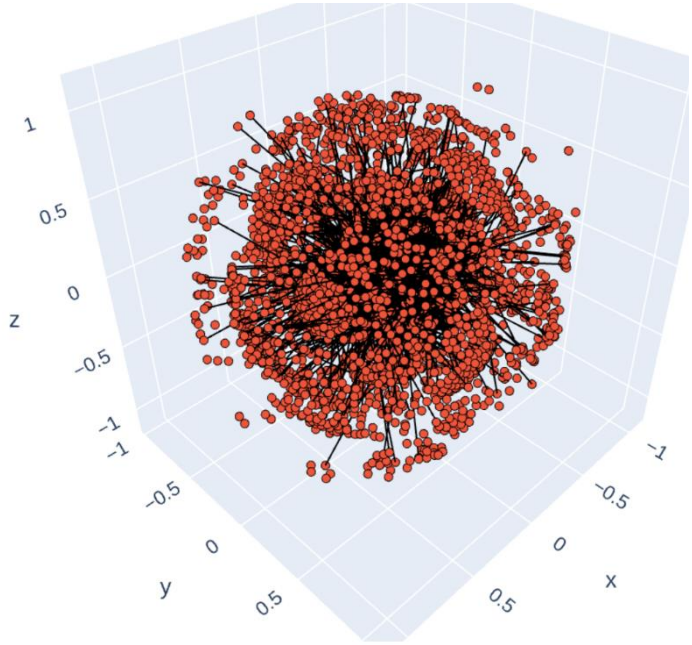


Рис. 17 Граф «Інтернет 3000»

На рисунку 23 відображено модель інтернету з 3000 вершин. Алгоритм працює в середньому за 6.5 хвилин. Структура отриманого результату відповідає очікуваній.

5. ПОРІВНЯННЯ

У цьому розділі проведено порівняння швидкості та ефективності алгоритмів KamadaNN з NeuLay-2 та Камада-Каваї на різних типах графів.

5.1. Симетрія

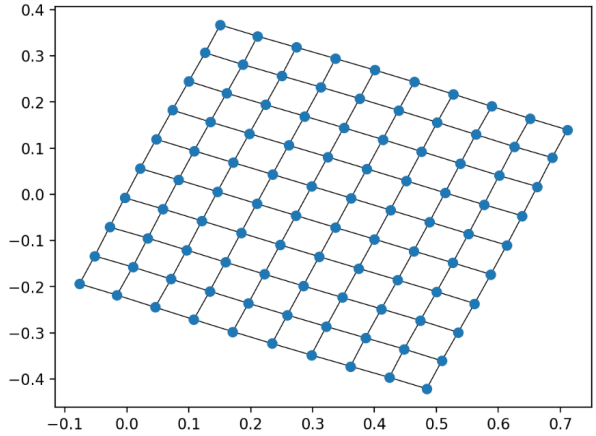


Рис. 18 Граф «Сітка 10 на 10», алгоритм KamadaNN

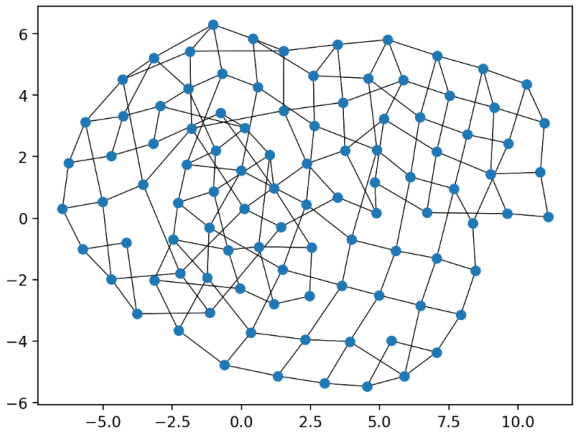


Рис. 19 Граф «Сітка 10 на 10», алгоритм NeuLay-2

На рисунках 18 та 19 проведено порівняння ефективності алгоритмів для графу сітки 10 на 10. Обидві програми працюють за 9-11 секунд, але NeuLay-2 видає доволі заплутаний результат за цей час.

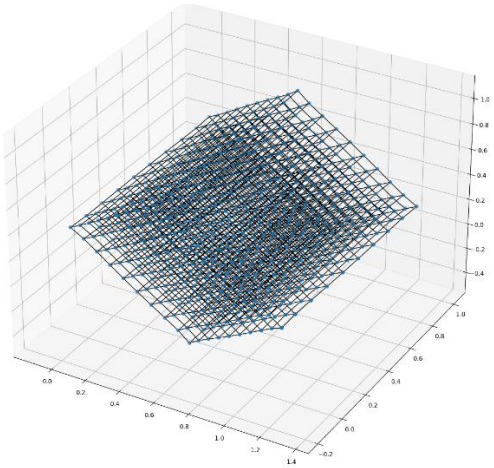


Рис. 20 Граф «Куб 10 на 10 на 10», алгоритм KamadaNN

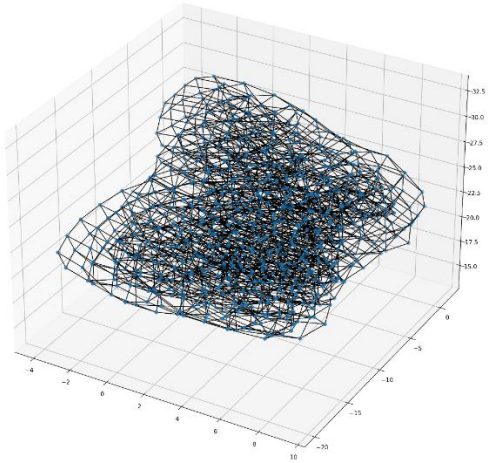
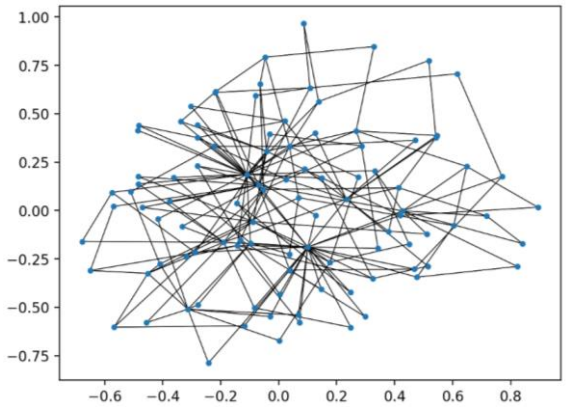


Рис. 21 Граф «Куб 10 на 10 на 10», алгоритм NeuLay-2

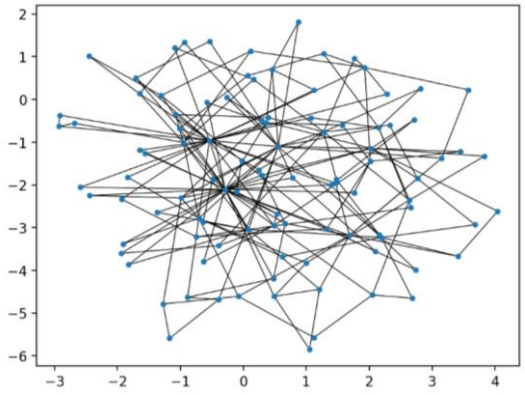
На рисунках 20 та 21 можна бачити такий самий результат порівняння на прикладі кубу 10 на 10 на 10. Обидві програми працюють в середньому 50 секунд, але NeuLay-2 не встигає розплутати граф.

5.2. Граф Барабаши

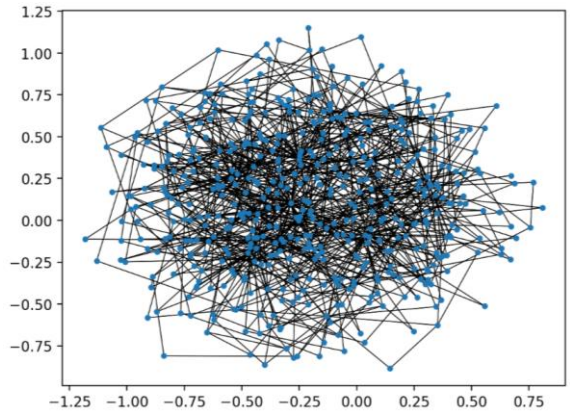
В статті Альберта Барабаши [10] було проведено багато експериментів саме для графа Барабаши, отже порівняємо алгоритми саме на такому графі.



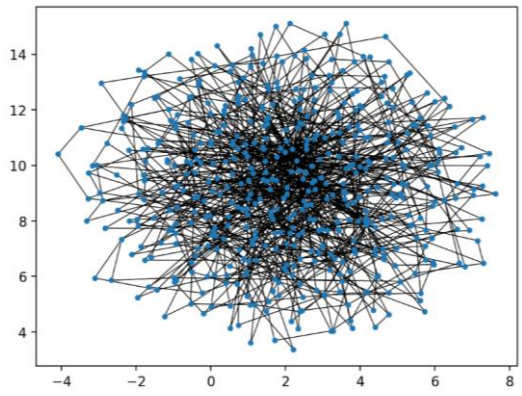
а) Kamada_NN, N=100



б) NeuLay-2, N=100



в) Kamada_NN, N=500



г) NeuLay-2, N=500

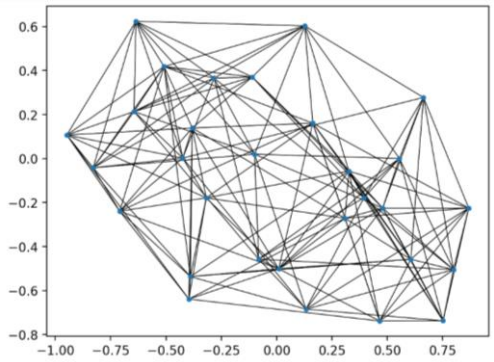
Рис. 22 Граф Барабаши

На рисунках 22 а-г можна бачити порівняння алгоритмів на 100 та 500 вершинах. Програми працюють приблизно однаковий час: 10-12 секунд для 100 вершин та 30-40 секунд для 500 вершин. Результат NeuLay-2 на 500 вершинах мені здається більш привабливим, ніж результат Kamada_NN, але варто зазначити, що це одна з проєкцій 3d графу у 2d простір, тому з іншого ракурсу це може виглядати інакше.

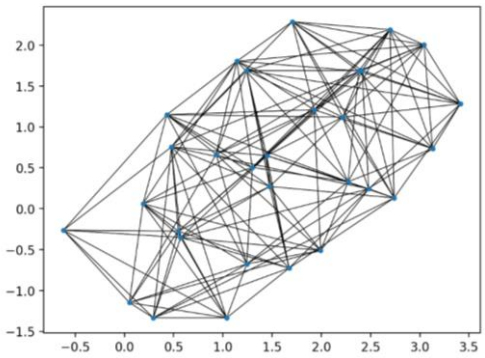
5.3. Граф Воттса-Строгаца

У цьому підрозділі вищезазначені алгоритми порівнюються в своїй ефективності на графах Воттса-Строгаца з різною кількістю вершин (N) та

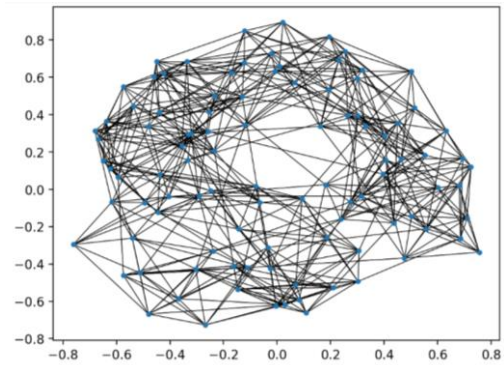
сусідів (K). Ймовірність переорієнтації ребра залишена автоматичною.



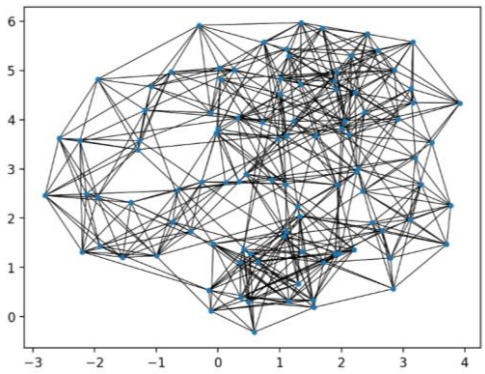
а) Kamada_NN, N=30, K=10



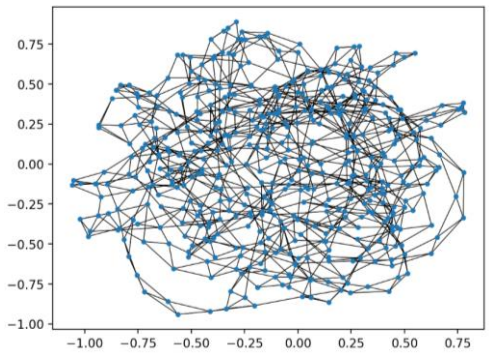
б) NeuLay-2, N=30, K=10



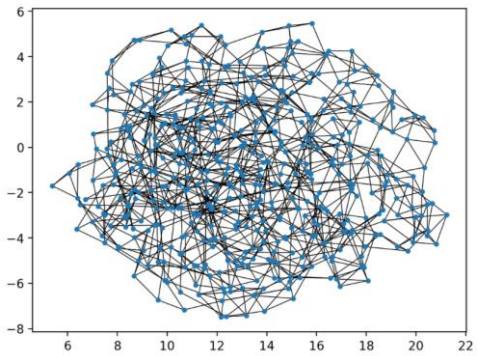
в) Kamada_NN, N=100, K=10



г) NeuLay-2, N=100, K=10



д) Kamada_NN, N=500, K=5



е) NeuLay-2, N=500, K=5

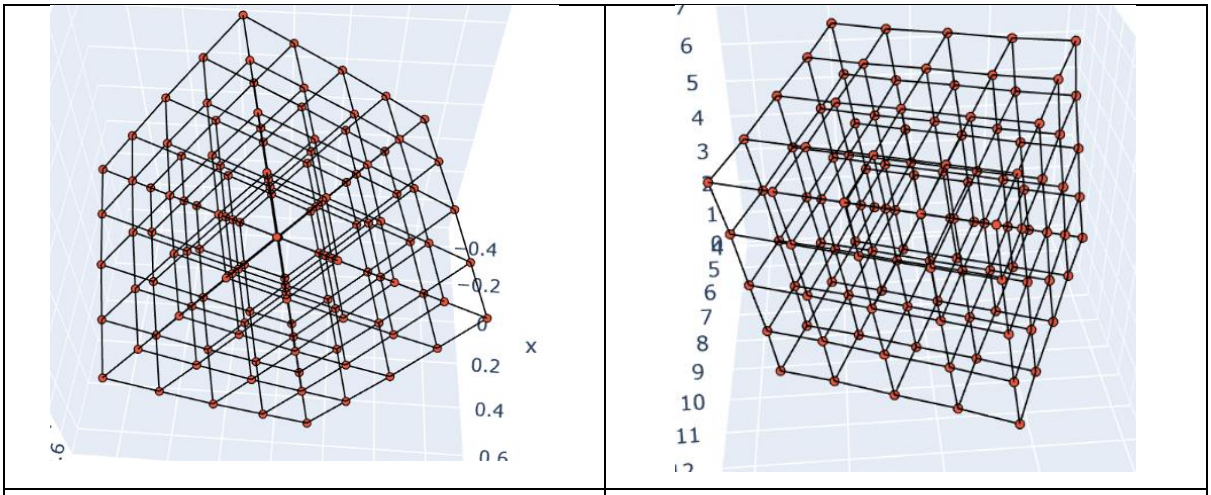
Рис. 23 Граф Вottса-Строгаца

Результати виконання обох алгоритмів наведено на малюнку 23. Ефективність обох алгоритмів на таких графах приблизно однакова. Структури дійсно нагадують маленькі містечка.

5.4. Порівняння KamadaNN та Камада-Каваї

Основною метою моєї роботи було прискорити алгоритм Камада-Каваї за допомогою нейронних мереж, отже проведемо порівняння цього

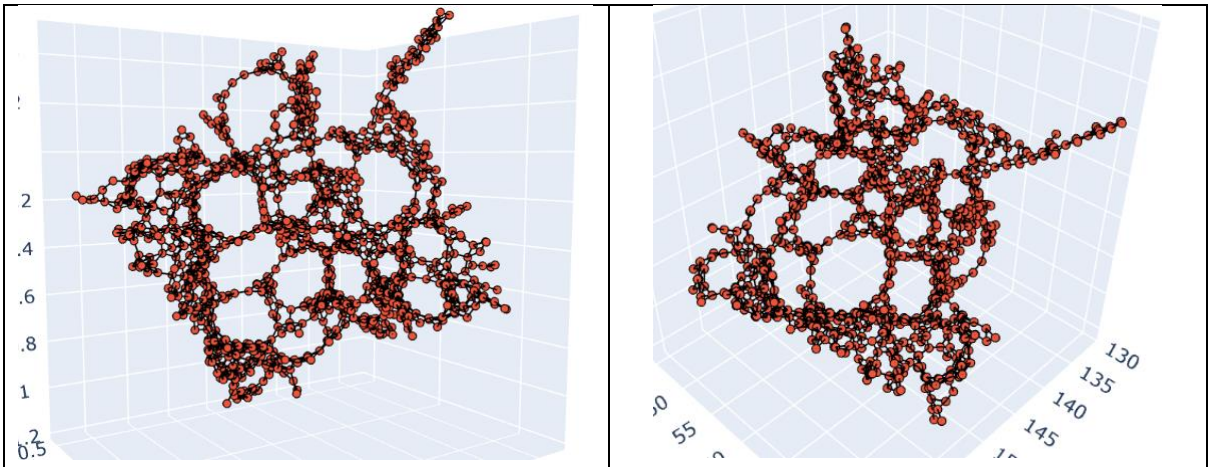
алгоритму та Kamada_NN.



а) Kamada_NN | *б) Камада-Каваї*

Рис. 24 Граф «Куб 5 на 5 на 5 вершин»

На рисунках 24а та 24б проведено порівняння на графі кубу зі 125 вершин. Kamada_NN працює в середньому 4 секунди, а Камада-Каваї 1 секунду. Структуру куба обидва алгоритми передають гарно.



а) Kamada_NN | *б) Камада-Каваї*

Рис. 25 Граф «Випадкова геометрія 1000 вершин»

Також було проведенно порівняння на графі випадкової геометрії з 1000 вершин (рис. 25а та 25б). Kamada_NN виконується за 47 секунд, Камада-Каваї за 4 хвилини 37 секунд.

Очікувано, Kamada_NN працює набагато швидше за Камада-Каваї на великих графах, та повільніше на маленьких.

ВИСНОВКИ

У роботі було:

1. Проведено огляд різних алгоритмів укладки графів. (розділ 2.2)
2. Зроблено огляд оптимізації силового алгоритму укладки графів за допомогою нейронних мереж (алгоритм Барабаши).
3. За допомогою підходу Барабаши розроблено та впроваджено оптимізовану версію алгоритму Камада-Каваї – KamadaNN (розділ 3).
4. Проведено велику кількість експериментів для дослідження ефективності та швидкості розробленого алгоритму. При проведенні експериментів враховувались симетрія, кластеризація та вагові показники отриманих укладок графів (розділ 4).
5. Проведено експерименти для порівняння алгоритмів KamadaNN та NeuLay-2 з точки зору швидкості та ефективності (розділ 5).

Результати експериментів показали, що запропонований алгоритм KamadaNN гарно відображає важливі для графів характеристики. Він демонструє здатність ефективно та доволі швидко зображувати графи різних структур та розмірів (розділ 4).

Виявлено, що алгоритм KamadaNN працює швидше в порівнянні з Камада-Каваї на великих графах та повільніше на маленьких. Також було виявлено, що в деяких аспектах, алгоритм KamadaNN працює краще за NeuLay-2, але в деяких гірше (розділ 5).

Для кращої візуалізації роботи алгоритму було додано нові методи зображення укладки графу за допомогою різних python-бібліотек (розділ 3.2).

Код програми наведено у [20].

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] J.A. Bondy, U.S.R. Murty. Graph Theory, Springer, 2008.

[2] M. Labonne. Hands-On Graph Neural Networks Using Python, <packt>, 2023.

[3] E. Kruja, J. Marks, A. Blair, R. Waters. A Short Note on the History of Graph Drawing, pp 272-286, in GD 2001, LNCS 2265, Springer-Verlag, Berlin, 2002.

[4] W. T. Tutte. How to Draw a Graph, pp 743-767, in Proceedings of the London Mathematical Society: Volume s3-13, Issue 1, 1963.

[5] https://igraph.org/r/doc/merge_coords.html

[6] T. Kamada, S. Kawai. An Algorithm for Drawing General Undirected Graphs, pp. 7-15, in Information Processing Letters 31, 1989.

[7] Г. М. Фіхтенгольц. Курс диференціального та інтегрального числення, том 1-3, М.: ФІЗМАТЛІТ, 2003.

[8] Cormen, Leiserson, Rivest, Stein. Introduction to Algorithms, 1990.

[9] T. Fruchterman, E. Reingold. Graph Drawing by Force-directed Placement, pp 1129-1164, in Software Practice and Experience, vol. 21(1 1), 1991.

[10] C. Both, N. Dehmamy, R. Yu, A. Barabási. Accelerating network layouts using graph neural networks, in Nature Communications, 2023.

[11] J. Barnes, P. Hut. A hierarchical O(N log N) force-calculation algorithm, pp. 446–449, in Nature. 324 (4), 1986.

[12] Документація бібліотеки *plotly*. https://plotly.com/python/

[13] Документація бібліотеки *matplotlib*. https://pypi.org/project/matplotlib/

[14] D. J. Watts, S. H. Strogatz. Collective dynamics of 'small-world' networks, pp. 440–442, in Nature. 393 (6684), 1998.

[15] Документація бібліотеки *NetworkX*. https://networkx.org/

[16] https://www.graphclasses.org/classes/gc_1341.html

[17] Документація бібліотеки PyTorch. https://pytorch.org/

[18] Документація бібліотеки PyTorch Geometric. https://pytorch-geometric.readthedocs.io/en/latest/

[19] Документація бібліотеки SciPy. https://scipy.org/

[20] Код KamadaNN: https://github.com/OlenaLinnyk/KamadaNN