Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики
Кафедра комп’ютерних наук та інформаційних систем

ДИПЛОМНА РОБОТА

на здобуття другого (магістерського) рівня вищої освіти
на тему
«Рекомендаційна система аналізу контенту баз даних та прийняття рішень»

Виконав: студент 6 курсу,
групи КНМ-21
спеціальності 122 «Комп’ютерні науки»
Кулик Юрій Васильович
Керівник: д.т.н. проф. Петришин
Любомир Богданович
Рецензент: к.т.н. доц. Горєлов Віталій
Олевтинович

Івано-Франківськ – 2021 р.

Державний вищий навчальний заклад

«Прикарпатський національний університет

імені Василя Стефаника»

Факультет математики й інформатики

Кафедра комп’ютерних наук та інформаційний технологій

Освітньо-кваліфікаційний рівень магістра

Спеціальність 122 Комп’ютерні науки

Затверджено на засіданні

кафедри_____

Протокол №_____ від_____

Завідувач кафедри

Петришин Л. Б. _____

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кулику Юрію Васильовичу

1. Тема роботи: Рекомендаційна система аналізу контенту баз даних та прийняття рішень

Керівник роботи: д.т.н. проф. Петришин Л.Б.

2. Перелік питань, які потрібно проаналізувати: аналіз та дослідження рекомендаційної системи

3. Дата видачі завдання: 23.09.2020

КАЛЕНДАРНИЙ ПЛАН

п/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1	Пошук джерел, наукових публікацій та статей.	02.11.2020 – 12.12.2020	
2	Вибір технологій для розробки.	13.12.2020 – 25.12.2020	
3	Дослідження бази даних Neo4j	26.12.2020 – 10.01.2021	
4	Пошук даних для дослідження	11.01.2021 – 05.02.2021	
5	Аналіз tmdb5000	06.02.2021 – 12.02.2021	
6	Реалізація імпорту даних з tmdb5000 у вигляд графу	13.02.2021 – 30.04.2021	
7	Пошук алгоритмів для кластеризації	01.05.2021 – 25.07.2021	
8	Аналіз існуючих алгоритмів для графів	25.07.2021 – 25.11.2021	
9	Здача на кафедру	11.12.2021	

Студент _____ Кулик Ю. В.

(підпис)

Керівник роботи _____ Петришин Л.Б.

(підпис)

АНОТАЦІЯ

«Рекомендаційна система аналізу контенту баз даних та прийняття рішень»

Кулик Ю. В., студент, спеціальність «Комп’ютерні науки та інформаційні технології»

Доктор технічних наук, доцент Петришин Л. Б.

Івано-Франківськ – 2021 р.

В даній роботі проаналізовано існуючі способи рекомендацій контенту, а також дослідження рекомендації на основі графової бази даних і кластеризації

Дана дипломна робота містить 76 сторінок, 16 рисунків, 16 формул, 5 таблиць та 46 джерел інформації.

ANNOTATION

«Recommended system of database content analysis and decision making»

Kulyk Y., student, specialty «Computer sciences and information technologies»

Prof., Ph.D. Petryshyn L.

Ivano-Frankivsk – 2021

This paper describes the existing methods of content recommendations, as well as the study of recommendations based on graph database and clustering

This thesis contains 76 pages, 16 drawings, 16 formulas, 5 tables and 46 sources of information

ЗМІСТ

ВСТУП

Ця робота присвячена розробці рекомендаційних систем для фільмів з використанням кластеризації баз даних графів. Метою даної роботи було дослідити використання графових баз даних як методу розробки системи рекомендацій фільмів.

Із зростанням кількості переглядів онлайн-фільмів корисно розробити систему, яка може надавати рекомендації користувачам [10]. Тут був використаний новий підхід із залученням графових баз даних. Графові бази даних — це метод з'єднання даних за допомогою теорії графів. З'єднання частин даних, які називаються вузлами, із зв'язком між ними дозволяє створити структуру, яку можна використовувати для зберігання, запитів і представлення даних [25].

У цій роботі досліджувалися унікальні властивості графових баз даних, які відокремлюють їх від різних типів баз даних. Було досліджено, як ці властивості можна використати для розробки системи, яка може рекомендувати користувачам фільми на основі фільмів, які їм раніше подобалися. Систему, що використовує базу даних такого типу, ще потрібно впровадити в комерційну систему, тому це область розробки рекомендаційної системи, яка має потенціал для багатьох подальших досліджень.

У цьому дослідженні спеціально розглядається властивість бази даних графів, яка називається кластеризацією. Оскільки дані представлені геометрично, можна знайти фрагменти даних, які згруповані разом у кластери, використовуючи багато різних алгоритмів. Існує гіпотеза, що при кластеризації даних фільмів будуть виявлені групи подібних фільмів. Ця робота намагається відповісти на два запитання: чи можна успішно представити базу даних, що містить дані про фільми, як базу даних графів? Чи можна використовувати алгоритми кластеризації графів для пошуку наборів фільмів, які можна використовувати як рекомендаційну систему?

Область математики, до якої належать бази даних графів, називається теорією графів. Протягом своєї історії було розроблено багато алгоритмів для

кластерних графіків. Незважаючи на те, що алгоритми були розроблені для виявлення кластеризації, наразі немає дослідження здатності цих алгоритмів знаходити кластери, які можна використовувати на даних фільмів для створення системи рекомендацій. Насправді, дослідження впливу цих алгоритмів на дані фільмів взагалі не проводилося. Поки невідомо, як будуть структуровані кластери, знайдені за допомогою цих алгоритмів [34].

Хоча графові бази даних використовувалися для розробки рекомендаційних систем, не було опублікованих досліджень щодо використання кластеризації як техніки.

Графові бази даних виявилися дуже корисними в комерційному світі. Всесвітня мережа — це графічна база даних. Розвиток Інтернету в цей граф був важливим для повсякденного використання, пошукова система Google базується на алгоритмі графу, який називається рейтингом сторінок [35]. Використання тут алгоритмів графів було революційним для того, як функціонує суспільство. Зрозуміло, що графові бази даних є потужним інструментом в інформаційному світі, і тому варто дослідити використання баз даних графів, теорії графів і алгоритмів графів з метою розробки рекомендаційної системи.

На поточні рекомендаційні системи, розроблені великими сервісами потокового відео, такими як Amazon та Netflix, було вкладено багато часу та грошей. Наявні системи працюють, рекомендуючи фільми на основі схожості інших користувачів, це називається спільною фільтрацією [36]. Основна відмінність від дослідженого тут підходу полягає в тому, що рекомендація ґрунтується виключно на власних уподобаннях користувачів. Ще одна проблема, яку можна вирішити за допомогою поточної системи рекомендацій, полягає в тому, що в міру появи нових фільмів базу даних можна негайно оновлювати та повторно кластерізувати, що дозволяє легко рекомендувати їх.

Однак спільна фільтрація повинна дати час, щоб новий фільм побачило достатньо людей, щоб дати значущу рекомендацію. Це називається проблемою холодного пуску. Оскільки це поширена проблема в

рекомендаційних системах, важливо, щоб дослідження було проведено для найкращого врахування цього [32]. Отже, у цьому дослідженні є багато переваг.

РОЗДІЛ 1. РЕКОМЕНДАЦІЙНІ СИСТЕМИ.

1.1 Рекомендаційні системи

Системи рекомендацій [23] отримали широке визнання та викликали підвищений інтерес громадськості протягом останнього десятиліття, прокладаючи шлях для нових можливостей продажу в інтернет комерції [35]. Наприклад, інтернет-магазини, такі як Amazon.com, успішно використовують широкий спектр різних типів рекомендаційних систем.

Їх головна мета полягає в тому, щоб зменшити складність для людини, профільтрувати дуже великі набори інформації та вибрати ті фрагменти ті фрагменти, які є релевантними для активного користувача. Крім того, рекомендаційні системи застосовують методи персоналізації, враховуючи, що різні користувачі мають різні переваги та різні інформаційні потреби [35]. Наприклад, якщо припустити, що це сфера рекомендацій щодо книг, то історики нібито більше зацікавляться середньовічною прозою, наприклад, Кентерберійськими оповіданнями Джефрі Чосера, ніж літературою про самоорганізацію, яка може бути більш актуальною для дослідників штучного інтелекту.

1.1.1 Збір інформації про переваги

Таким чином, щоб генерувати персоналізовані рекомендації, адаптовані до конкретних потреб активного користувача, системи рекомендацій повинні збирати інформацію про особисті переваги, наприклад, історію покупок користувача, дані кліків, демографічну інформацію, тощо. Традиційно вирази переваг користувачів a_i, щодо продуктів b_k називають рейтингами $r_i(b_k)$. Розрізняють два різних типи рейтингів:

Відверті оціни. Користувачі зобов'язані чітко вказати свої переваги щодо будь-якого конкретного предмета, зазвичай, вказуючи ступінь оцінки за 5-бальною або 7-бальною шкалою лайкерта. Потім ці шкали перетворюються на числові значення, наприклад безперервні діапазони $r_i(b_k) \in [-1, +1]$. Негативні значення зазвичай вказують на неприязнь, тоді як позитивні значення виражають симпатію користувача.

Неявні оцінки. Відверті оцінки доставляють користувачам додаткові зусилля. Таким чином, користувачі часто намагаються уникати тагаря прямого вказання своїх уподобань і або залишають систему, або покладаються на “те що буде”. З іншого боку, отримання інформації про переваги з простих спостережень за поведінкою користувачів є менш нав’язливим. Типовим прикладом неявних рейтингів є дані про покупку, час читання новин та поведінка при перегляді. Незважаючи на те, що їх легше зібрати, неявні рейтинги мають серйозні наслідки. Наприклад, деякі покупки є подарунками і, таким чином, не відображають інтереси активного користувача. Більше того, висновок про те, що покупка означає симпатію, не завжди справедливий.

Через труднощі з отриманням чітких рейтингів, деякі постачальники послуг з рекомендаційних товарів використовують двосторонні підходи. Наприклад Amazon.com вичисляє рекомендації на основі явних оцінок, коли це можливо. У разі недоступності замість них використовуються спостережувані неявні оцінки.

1.1.2 Типи та методи рекомендаційної системи

Виникли дві основні парадигми для обчислювальних рекомендацій, а саме фільтрація на основі вмісту та спільна фільтрація, або колаборативна фільтрація [28]. Фільтрація на основі вмісту, яку також називають когнітивною фільтрацією [24], обчислює подібність між кошиком цінних продуктів активного користувача a_i та продуктами зі всього додатку продуктів, які досі невідомі для a_i. Схожість продукту визначається за допомогою вибраних атрибутів на властивостями цього продукту. Тоді як спільна фільтрація, яку також називають соціальною фільтрацією, обчислює схожість між користувачами на основі їх рейтингового профілю. Більшість подібних користувачів потім служать “порадниками”, пропонуючи найбльш релевантні продукти активному користувачеві.

Розширені системи рекомендацій, як правило, поєднують спільну фільтрацію та фільтрацію на основі вмісту, намагаючись пом’якшини

недоліки будь-якого підходу та використовувати синергетичні ефекти. Ці системи отримали назву “гібридні системи” [12].

1.2 Фільтрація на основі вмісту

Підходи до складання рекомендацій, засновані на змісті, глибоко вкорінені в пошук інформації [36]. Як правило, ці системи використовують байєсівські класифікатори за допомогою функцій змісту, або виконують запити у векторному просторі найближчого сусіда. Байєсівські класифікатори використовують теорему Байєса про умовну незалежність [36]:

$$P(R|F) = \frac{P(F|R) \cdot P(R)}{P(F)} \quad (1.1)$$

Більше того, байєсівські класифікатори роблять “наївне” припущення, що ознаки опису продукту є незалежними, що зазвичай не так. Враховуючи мітку класу, ймовірність незалежності b_k до класу R_i враховуючи його n значень ознак F_1. . ., F_n, визначається наступним чином:

$$P(R_i, \ldots, F_n) = \frac{1}{Z} \cdot P(R_i) \cdot \prod_{j=1}^{n} P(F_j|R_i) \quad (1.2)$$

Змінна Z представляє коефіцієнт масштабування, що залежить лише від F_1,. . ., F_n. Ймовірності $P(R_i)$ і $P(F_j|R_i)$ можна оцінити за навчальними даними. Для запитів у векторному просторі атрибути, наприклад, терміни простого тексту або машиночитані метадані, витягуються з описів продуктів і використовуються для профілювання користувачів і представлення продукту. Наприклад Фаб [6] представляє документи в термінах 100 слів з найвищою вагою TF-IDF, тобто словами, які зустрічаються частіше в цих документах, ніж у середньому.

1.3 Спільна фільтрація

Фільтрація на основі вмісту працює лише у тих доменах, де вилучення ознак можливе, а інформація про атрибути легкодоступна. Спільна фільтрація, з іншого боку, використовує уявлення без вмісту і не стикається з

тими самими обмеженнями. Наприклад, Джестер [38], використовує спільну фільтрацію, щоб рекомендувати жарти своїм користувачам. У той час як фільтрація на основі вмісту враховує описові характеристики продуктів, спільна фільтрація використовує оцінки, які користувачі надають продуктам. Отже, алгоритми колаборативної фільтрації зазвичай працюють з набором користувачів $A = \{a_1, a_2, \ldots, a_n\}$, набором продуктів $B = \{b_1, b_2, \ldots, b_m\}$ і функціями часткового оцінювання $r_i : B \rightarrow [-1, +1]\perp$ для кожного користувача $a_i \in A$. Від'ємні значення $r_i(b_k)$ означають неприязнь, а позитивні значення виражають симпатію a_i до продукту b_k. Нижні значення $r_i(b_k) = \perp$вказують на те, що a_i на має рейтингу b_k.

Завдяки високій якості продукції та мінімальним вимогам до інформації, системи спільної фільтрації стали найбільш яскравими представниками рекомендаційних систем. Багато комерційних постачальників, наприклад, Amazon.com [39], використовують варіації методів колаборативної фільтрації, щоб пропонувати продукти своїм клієнтам. Крім простих байєсівських класифікаторів [36], хортинг та методики, засновані на правила асоціації, переважно два підходи набули широкого поширення, а саме, заснований на користувачах і на елементах на основі спільної фільтрації. Насправді, термін "спільна фільтрація" зазвичай використовується як синонім до "спільна фільтрація на основі користувачів", завдяки величезній популярності цієї техніки. Наступні два розділи приблизно описують алгоритмічні реалізації спільної фільтрації як на основі користувачів, так і на основі елементів.

1.3.1 Спільна фільтрація на основі користувачів

Проекти Ringo та GroupLens [40] були одними з перших рекомендаційних систем, які застосовували методи, відомі як "спільна фільтрація на основі користувачів". Представляючи функцію оцінки a_i кожного користувача r_i як вектор, вони спочатку обчислюють схожість $c(a_i, a_j)$ між усіма парами $(a_i, a_j) \in A \times A$. З цією метою використовуються загальні статистичні коефіцієнти кореляції [9], як правило, кореляція Пірсона і

косинусна міра подібності, добре відома з пошуку інформації [8]. Як випливає з назви, косинусна міра подібності кількісно визначає подібність двох векторів $\vec{v}_i$ $\vec{v}_j$ $[-1, +1]^{|B|}$ за косинусом їхніх кутів:

$$sim(\vec{v}_i, \vec{v}_j) = \frac{\sum_{k=0}^{|B|} v_{i,k} \cdot v_{j,k}}{\left(\sum_{k=0}^{|B|} v_{i,k}^2 \cdot \sum_{k=0}^{|B|} v_{j,k}^2\right)^{\frac{1}{2}}} \quad (1.3)$$

Кореляція Пірсона, отримана з моделі лінійної регресії, подібний до косинусної подібності, але вимірює ступінь лінійної залежності яка існує між двома змінними. Символи $\overline{v_i}$, $\overline{v_j}$ позначають середні значення векторів $\vec{v}_i$, $\vec{v}_j$

$$sim(\vec{v}_i, \vec{v}_j) = \frac{\sum_{k=0}^{|B|} (v_{i,k} - \overline{v_i}) \cdot (v_{j,k} - \overline{v_j})}{\left(\sum_{k=0}^{|B|} (v_{i,k} - \overline{v_i})^2 \cdot \sum_{k=0}^{|B|} (v_{j,k} - \overline{v_j})^2\right)^{\frac{1}{2}}} \quad (1.4)$$

Для обчислення подібності $c(a_i, a_j) \in A \times A$ за допомогою косинусної міри подібності або кореляції Пірсона, будуються околиці $prox(a_i)$ найбільш подібних сусідів top-M для кожного рівного $a_i \in A$. Далі обчислюються передбачення для всіх добутків b_k, які оцінили сусіди a_i, але які ще невідомі a_i, тобто, більш формально, передбачення $w_i(b_k)$ для $b_k \in \{b \in B \mid \exists\ a_j \in prox(a_i) : r_j(b) \neq \perp\}$:

$$\omega_i(b_k) = \overline{r_i} + \frac{\sum_{a_i \in prox(a_i)} (r_i(b_k) - \overline{r_j}) \cdot c(a_i, a_j)}{\sum_{a_i \in prox(a_i)} c(a_i, a_j)} \quad (1.5)$$

Таким чином, передбачення базуються на середньозважених значення відхилень від середніх значень сусідів a_i. Для топ-N рекомендацій обчислюється список $P_{wi} : \{1,2,...,N\} \rightarrow B$ на основі передбачення w_i.

Зауважте, що функція P_{wi} є ін'єктивною і відображає рейтинг рекомендацій у порядку спадання, першими даючи найвищі прогнози.

Щоб зробити кращі прогнози, різні дослідники запропонували кілька модифікацій основного алгоритму спільної фільтрації на основі користувачів. Нижче наведено найвідоміші з них, але, безумовно, їх є набагато більше.

- Інверсна частота користувача. У програмах пошуку інформації, заснованих на моделі векторного простору, частоти слів зазвичай змінюються за фактором, відомим як "інверсна частота документа" [38]. Ідея полягає в тому, щоб зменшити вплив слів, що часто зустрічаються і збільшити вагу для незвичайних термінів під час обчислення подібності між векторами документа. Зворотна частота користувачів, вперше згадана Breese приймає це поняття і винагороджує співголосування за менш поширені товари набагато більше, ніж співголосування за дуже популярні продукти.
- Зважування значимості. Обчислення кореляцій користувача та користувача $c(a_i, a_j)$ враховує лише продукти, які обидва користувачі оцінили, тобто $b_k \in (\{b \mid r_i(b) \neq \perp\} \cap \{b \mid r_j(b) \neq \perp\})$. Таким чином, навіть якщо a_i та a_j мають кооперацію лише з один добутком b_k, вони матимуть максимальну кореляцію, якщо виконується $r_i(b_k) = r_j(b_k)$. Очевидно, що такі кореляції, засновані лише на кількох точках даних, є не дуже надійними. Герлокер запропонував штрафувати кореляції користувачів на основі менш ніж 50 загальних оцінок, застосовуючи вагу значимості s/50, де s позначає кількість предметів спільної оцінки. Голосування за замовчуванням є ще одним підходом до вирішення тієї ж проблеми [30].
- Корпус посилення. Хоча обидві попередні модифікації стосуються подібних процесів обчислення, посилення випадку розглядає крок прогнозування рейтингу [38], формалізованого у рівнянні 2.5. Підкреслюються коефіцієнти кореляції $c(a_i, a_j)$, близькі до одиниці, а низькі коефіцієнти кореляції караються:

$$c'(a_i, a_j) = \begin{cases} c(a_i, a_j)^{\rho}, & c(a_i, a_j) \geq 0 \\ -(-c(a_i, a_j))^{\rho} \end{cases} \quad (1.6)$$

Отже, дуже схожі користувачі мають набагато більший вплив на прогнозовані рейтинги, ніж раніше. Зазвичай ρ приймає значення близько 2,5.

Деякі дослідники [40] дещо розширили концепцію спільної фільтрації на основі користувачів і додали фільтр-ботів як додаткових користувачів, які можуть бути обрані як сусіди для "справжніх" користувачів. Filterbots - це автоматизовані програми, які поводяться певним, заздалегідь визначеним чином. Наприклад, у контексті рекомендації новин, деякі фільтр боти оцінювали татті Usenet на основі частки орфографічних помилок, тоді як інші зосереджувалися на довжині тексту тощо. Sarwar показа, що фільтр-боти можуть підвищити точність рекомендацій при роботі в малонаселених системах спільної фільтрації.

1.3.2 Спільна фільтрація на основі елементів

Спільна фільтрація на основі елементів [23] набирає обертів протягом останніх п'яти років завдяки сприятливим характеристикам складності обчислень і здатності відокремити процес моделювання обчислень від реального прогнозування. Особливо для випадків коли $|A| \gg |B|$, було показано, що обчислювальна продуктивність спільної фільтрації на основі елементів перевищує спільну фільтрацію на основі користувачів. Його успіх також поширюється на багато комерційних рекомендаційних систем, таких як Amazon.com [29].

Як і спільна фільтрація на основі користувачів, створення рекомендацій базується на оцінках $r_i(b_k)$, які користувачі $a_i \in A$ надають для продуктів $b_k \in B$. Однак, на відміну від спільної фільтрації на основі користувачів, значення подібності c обчислюються для елементів, а не для користувачів, отже $c : B \times B \rightarrow [-1, +1]$. Грубо кажучи, два елементи b_k, b_e подібні, тобто мають велике значення $c(b_k, b_e)$, якщо користувачі, які оцінюють один з них, мають

тенденцію оцінювати інший, і якщо користувачі схильні призначати їм ідентичні або подібні оцінки. Фактично, обчислення подібності на основі елемента прирівнюється до випадку на основі користувача при повороті матриці продукт-користувач на 90 градусів. Даля для кожного b_k визначаються околиці $prox(b_k) \subseteq B$ найдрібніших елементів топ-M $w_i(b_k)$ обчислюються наступним чином:

$$w_i(b_k) = \frac{\sum_{b_e \in B'_k} (c(b_k, b_e) \cdot r_i(b_e))}{\sum_{b_e \in B'_k} |c(b_k, b_e)|} \quad (1.7)$$

Інтуїтивно, цей підхід намагається імітувати реальну поведінку користувача, коли користувач a_i оцінює цінність невідомого продукту b_k, порівнюючи останній із відомими, подібними елементами та враховуючи, наскільки вони цінуються.

Остаточне обчислення списку P_{wi} з перших N рекомендацій слідує процесу спільної фільтрації на основі користувача, упорядковуючи рекомендації відповідно до w_i в порядку спадання.

1.4 Гібридні рекомендаційні системи

Гібридні підходи спрямовані на уніфікацію спільної фільтрації та фільтрації на основі вмісту в рамках однієї системи, використовуючи синергетичні ефекти та пом'якшуючи недоліки, притаманні кожній із парадигм. Отже, гібридні рекомендації використовують як інформацію про рейтинг продукту, так і описові характеристики. Насправді можна уявити безліч способів поєднання аспектів співпраці та змісту, Берк перераховує безліч методів гібридизації. Найбільш широко поширеною серед них, однак, є так звана парадигма [27] “співпраці через контент”, коли профілі на основі вмісту створюються для виявлення подібності між користувачами.

Однією з найперших гібридних рекомендаційних систем є Fab [26], яка пропонує своїм користувачам веб-сторінки. Melville та Hayes and Cunningham використовують інформацію про вміст для прискорення процесу спільної фільтрації. Торрес пропонує різні гібридні системи для рекомендації

цитування наукових робіт. Хуанг використовує функції на основі вмісту, щоб побудувати графіки кореляції для дослідження транзитивних асоціацій між користувачами.

Модельно-керовані гібридні підходи ули запропоновані Basilico та Hofmann пропонуючи навчання персептрону та функції ядра, а також Шейн використовуючи більш традиційні байєсівські класифікатори.

1.5 Оцінка рекомендаційних систем

Оцінки рекомендаційних систем є незамінними для того, щоб кількісно оцінити, наскільки корисні рекомендації, зроблені системою S_x, порівнюються з S_y для повного набору користувачів A. Онлайн оцінки, тобто пряме запитання користувачів щодо їхньої думки, у більшості випадків не являється вибором.

Причин багато, наприклад:

Розгортання. Для виконання онлайн-оцінок необхідна цілісна віртуальна спільнота, здатна запускати служби рекомендаційної системи. З іншого боку, успішне розгортання онлайн-спільноти та забезпечення її самопідтримки є громіздким і може вийти за межі більшості дослідницьких проектів.

Нав’язливість. Навіть якщо онлайн-спільнота є легкодоступною, оцінювання не можна просто виконувати за бажанням. Багато користувачів можуть розглядати анкети як додатковий тягар, що не дає їм негайної винагороди, і, можливо, навіть вирішить залишити систему.

Отже, дослідження в основному спиралися на офлайнові методи оцінки, які застосовні до наборів даних, що містять попередні рейтинги продуктів, таких як, наприклад, добре відомі набори даних MovieLens і Every Movie, обидва загальнодоступні. Методи машинного навчання які називаються крос валідація застосовуються до цих наборів даних, наприклад, утримання, К-згортання, або тестування без урахування, а також для метрики оцінки. У наступних розділах наведено опис популярних показників, які

використовуються для оцінки в автономному режимі. Розширене та більш повне опитування надано Herlocker [41].

1.5.1 Показники точності

Показники точності були визначені насамперед для двох основних завдань: по-перше, щоб судити про точність окреми передбачень, тобто наскільки прогнози $w_i(b_k)$ для продуктів b_k відхиляються від фактичних рейтингів a_i $r_i(b_k)$. Ці показники особливо підходять для завдань, де передбачення відображаються разом із продуктом, наприклад анотація в контексті [41]. По-друге, метрики підтримки прийняття рішень оцінюють ефективність допомоги користувачам у виборі високоякісних елементів із набору всіх продуктів, загалом припускаючи бінарні переваги.

1.5.1.1 Показники точності прогнозування

Показники точності прогнозування визначають, наскільки близькі прогнозовані оцінки до реальних оцінок користувачів. Найбільш відомі та широко використовувані [41], середня абсолютна помилка (MAE) являє собою ефективний засіб для вимірювання статистичної точності прогнозів $w_i(b_k)$ для множин B_i продуктів:

$$|\overline{E}| = \frac{\sum_{b_k \in B_i} |r_i(b_k) - \omega_i(b_k)|}{|B_i|} \tag{1.8}$$

Пов’язана з MAE, середня квадратична помилка (MSE) квадратує помилку перед підсумовуванням. Таким чином, великі помилки стають набагато виразнішими, ніж малі. Дуже прості в реалізації показники точності прогнозування не підходять для оцінки якості топ-N списків рекомендацій: користувачів хвилюють лише помилки для продуктів високого рангу. З іншого боку, помилки передбачення для продуктів низького рангу не мають значення, оскільки користувач все одно не цікавиться ними. Однак MAE і MSE враховують обидва типи помилок абсолютно одинаково.

1.5.1.2 Показники підтримки рішень

Точність і відкликання, добре відомі з пошуку інформації, не враховують прогнози та їх відхилення від реальних рейтингів. Вони радше судять про те, наскільки актуальними для активного користувача є набір рейтингових рекомендацій. Як правило, перед використанням цих показників застосовується K-згортання, поділяючи оцінені продукти кожного користувача a_i $b_k \in Ri = \{b \in B | r_i(b) \neq \perp\}$ на K неперетинних фрагментів бажано однакового розміру. Зазвичай припускаються параметри складання $K \in \{4,5,...,10\}$. Далі K-1 випадково вибраних зрізів використовується для формування навчальної множини R_i^x для a_i. Ці рейтинги потім визначають профіль A_i, на основі якого обчислюються остаточні рекомендації. Для генерації рекомендацій залишковий зріз a_i ($R_i \setminus R_i^x$) зберігається і не використовується для передбачення. Цей зріз позначений T_i^x, становить тестовий набір, тобто ті продукти, які планують передбачити рекомендаційні алгоритми.

Точність, Відкликання та F1

Сарвар представляє адаптований варіант відкликання, фіксуючи відсоток продуктів тестового набору $b \in T_i^x$, що зустрічаються в списку рекомендацій P_i^x щодо загальної кількості продуктів тестового набору $|T_i^x|$.

$$\text{Відкликання} = 100 \cdot \frac{|T_i^x \cap \Im P_i^x|}{|T_i^x|} \tag{1.9}$$

Символ $\Im P_i^x$ позначає зображення карти P_i^x, тобто всі елементи списку рекомендацій.

Відповідно точність представляє відсоток продуктів тестового набору $b \in T_i^x$, що зустрічаються в P_i^x, по відношенню до розміру списку рекомендацій:

$$\text{Точність} = 100 \cdot \frac{|T_i^x \cap \Im P_i^x|}{|P_i^x|} \quad (1.10)$$

Інший популярний показник, який широко використовується в дослідженнях систем пошуку інформації та рекомендацій є стандартною метрикою F1. F1 поєднує точність і відкликання в одній метриці, надаючи їм однакову вагу:

$$F1 = \frac{2 \cdot \text{Відкликання} \cdot \text{Точність}}{\text{Відкликання} + \text{Точність}} \quad (1.11)$$

Оцінка Бріза

Бріз вводить цікаве розширення для відкликання, відоме як зважене відкликання або оцінка Бріза. Основна ідея відноситься до інтуїції, що очікувана корисність списку рекомендацій - це просто ймовірність перегляду рекомендованого продукту, який насправді є релевантним, тобто взятий з набору тестів, помножених на його корисність, яка дорівнює 0 або 1 для неявних оцінок. Крім того, Бріз стверджує, що кожен наступний елемент у списку менш ймовірно буде переглянутий активним користувачем з експоненційним спадом. Очікувана корисність рейтингового списку продуктів P_i^x виглядає наступним чином:

$$H(P_i^x, T_i^x) = \sum_{b \in (T_i^x \cap \Im P_i^x)} \frac{1}{2^{(P_i^x)^{-1}(b)-1/(a-1)}} \quad (1.12)$$

Параметр a позначає період напіврозпаду перегляду. Період напіврозпаду - це кількість продуктів в списку, при якій існує 50% ймовірність того, що активний агент, представлений навчальним наобром R_i^x, перегляне цей продукт. Нарешті, зважене відкликання P_i^x щодо T_i^x визначається наступним чином:

$$BScore(P_i^x, T_i^x) = 100 \cdot \frac{H(P_i^x, T_i^x)}{\sum_{k=1}^{|T_i^x|} \frac{1}{2^{(k-1)/(a-1)}}} \quad (1.13)$$

Цікаво, що при припущенні a = ∞ оцінка Бріза ідентична незваженому відкликанню.

Інші популярні показники підтримки прийняття рішень включаються ROC, так звана робоча характеристика приймача. ROC вимірює ступінь, до якого система фільтрації інформації здатна успішно розрізняти сигнал і шум. Менш части використовуваний NDPM порівнює два різних, слабко впорядкованих рейтинга.

1.5.1.3 За межами точності

Хоча показники точності є важливим аспектом корисності, є риси задоволеності користувачів, які вони не можуть охопити. Тим не менш, показники, що не є точними, до цього часу в основному не викликали серйозного дослідницького інтересу, і вони розглядалися лише як незначно важливі доповнення до показників точності.

1.5.1.3.1 Покриття

Серед усіх показників оцінки неточності, покриття було найбільш часто використовуваним [41]. Покриття вимірює відсоток елементів у проблемній області, для яких можна зробити прогноз.

Наприклад, припустивши підхід спільної фільтрації на основі користувачів, представлений у розділі 1.3.2.2.1, охоплення для всієї групи користувачів обчислюється наступним чином:

$$\text{Покриття} = 100 \cdot \frac{\sum_{a_i \in A} |\{b \in B | \exists\, a_j \in prox(a_i) : r_j(b) \neq \perp\}|}{|B| \cdot |A|} \quad (1.14)$$

1.5.1.3.2 Новизна і випадковість

Деякі рекомендації дають дуже точні результати, які все ще марні на практиці, наприклад, пропонують банани покупцям у продуктовому магазині: майже всі цінують банани, тому їхні рекомендації мають на увазі високу точність. З іншого боку, через їх високу популярність більшість людей інтуїтивно купують банани, зайшовши в продуктовий магазин [41]. Вони не потребують додаткової рекомендації, оскільки вони “вже знають”.

Таким чином показники новизни та випадковості вимірюють неочевидність зроблених рекомендацій, покарання за “збір вишні”.

РОЗДІЛ 2. ГРАФОВА БАЗА ДАНИХ. РЕКОМЕНДАЦІЙНА СИСТЕМА НА ОСНОВІ ГРАФІВ.

2.1 Графова база даних

Графова база даних - це NoSQL база даних, яка має структуру графу. Вона складається з вузлів, які представляють сутності даних які з'єднані між собою ребрами, які представляють відносини між цими об'єктами [2]. Графові бази даних надзвичайно корисні для сильно взаємопов'язаних даних і як такі викликали великий інтерес у галузях біології, інформатики та інформаційних технологій. Юн та співавтори, у статті 2017 року "Використання графових баз даних для інтеграції гетерогенних біологічних даних", успіх графових баз даних у складних біологічних проблемах[2]. Використання цієї технології в такій складній задачі показує силу, яку мають ці бази даних. Це стаття з великим цитуванням, яка показує справжній потенціал цієї технології.

Графова база даних показана на рисунку 2.1. Цей рисунок було взято з офіційного сайту Neo4j, велика компанія програмного забезпечення для графічних баз даних. Оскільки ця компанія є основним джерелом технології графових баз даних, вона є і корисним джерелом інформації про графові бази даних[3].

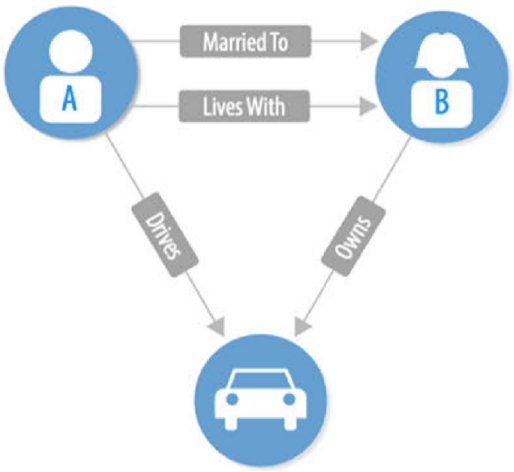

Рис. 2.1. Приклад графової бази даних

На рисунку 2.1 показано як дві людини A і B, пов’язані один з одним, і як обидва пов’язані з автомобілем. Перша пропозиція щодо графової бази даних як методу опису даних була в статті Джона Сови 1979 [43] року “Концептуальні графіки для інтерфейсу бази даних”. У цій статті концепція графічної бази даних розглядається як засіб опису даних, а не як зберігання. Граф мав бути посередником між людиною-користувачем і комп’ютером. Концепція була задумана як метод запиту комп’ютерної системи. Система повинна була перетворити питання з людської мови на концептуальний графік. Система може шукати інші графи в базі даних, які мають відношення до вихідного запитання.

З тих пір багато систем графових баз даних було розроблено комерційно. Першою комерційною графовою базою даних була Allegro graph у 2004 році, ця база даних спочатку була розроблена для зберігання трійок RDF [8]. Дуже помітним прикладом використання графової бази дани є Amazon Neptune, вперше обговорений у 2011 році в статті Кріса Бранча [9] “Neptune: мова, специфічна для домену для розгортання програмного забезпечення на хмарних платформах”. Система була випущена в 2018 році. Це розширення веб-сервісів Amazon. Amazon стверджує, що ця система є швидшим і ефективнішим способом запуску веб-додатків, які працюють з неймовірно великими наборами даних [22]. Хоча ця інформація взята з документації Amazon з невеликими доказами. Через те, що система нова, часу на створення повних світів про її можливості було мало. Багато інформації про Neptune є конфіденційною, оскільки цей продукт є великою інвестицією для Amazon.

Також було випущено багато поулярних систем графічних баз даних з відкритим вихідним кодом. Одним із популярних прикладів є Neo4j, програма запитів до графової бази даних на основі Java, .NET, JavaScript, Python, Ruby, яка була випущена в 2007 році [29]. Ця програма може бути використана для зберігання даних у форматі графа.

Також були розроблені мови запитів, щоб збільшити корисність графових баз даних, як методів зберігання даних. Поширеною мовою є Cypher.

2.1.1 Графові бази даних для рекомендаційних систем

У цій роботі було дослідження графове представлення як потенційний метод рекомендації для цифрових бібліотек. Було створено мережу графів між двома базами даних SQL. Перший рівень - це вміст книги, а другий - демографічні дані клієнтів. Як і у стандартних графових базах даних, точки даних розглядалися як вузли, а відносини як ребра [26]. Однах з'єднання вузлів могло відбуватися лише між двома шарами. Навіть з цим обмеженням, при оцінці людей було отримано похибку 18.3% і точність 38.1%. Хоча предметна оцінка цього дослідження була неймовірно обмеженою. Суб'єктами були лише два студенти.

Лише в середині кінця 2000-х років були створені справжні комерційні графові бази даних з такими пакетами як neo4j у 2010 році. Ефективність системи графової бази даних на відміну від стандартної реляційної бази даних була продемонстрована в статті "Chad Vicknair et al 2010 A Comparison" [44]. У статті досліджуються два типи запитів: структурні запити та запити даних. Запити даних: підрахувати кількість вузлів, дани корисного навантаження яких дорівнюють деякому значенню. Підрахувати кількість вузлів, дані корисного навантаження яких менші за деяке значення. Підрахувати кількість вузлів, дані корисного навантаження яких містять певний рядок пошуку (довжина коливається від 4 до 8). Результатом цього було те що Neo4j працював краще ніж MySQL, виконуючи запити швидше. Однак це не стосується конкретно рекомендаційних систем.

На початку 2010-х років було розроблено кілька мов запитів до графічних баз даних. У 2013 році Holzschuher. порівняли мови графічних баз даних Cypher, Gremlin та Neo4j у своїй роботі 2003 року "Продуктивність Графових мов запиту". Дані тесту містили 2011 осіб, 26982 повідомлення, 25365 видів діяльності, 2000 адрес, 200 груп і 100 організацій. Це значна

кількість даних, яка повинна стати хорошим тестом мови на основі графів. Отриманий результат показав, що пови графових баз даних були на порядок вищі ніж мови реляційних баз даних. Стосовно рекомендаційних систем, Frano is Fouss розробили спільну роботу на основі графових баз даних, “Експериментальне дослідження ядер графів на задачі спільної рекомендації”. У цій статті описано створення системи рекомендацій на основі даних з MovieLens. Ці дані складаються з людей, фільмів і категорій фільмів. Вузли цього графа були визначені людьми а ребра - відносинами. А саме “has_watched”, між людиною і фільмом, та “belongs_to” між фільмом та категорією. Це викликало кластеризацію вузлів у межах гарфа. Людям в кластері будуть рекомендовані фільми, які сподобалися іншим людям в тому ж кластері. Кластери визначають шляхом розрахунку евклідової відстані між вузлами. Результат цієї роботи показав, “що три заходи подібності забезпечують хорошу та стабільну роботу”.

Очевидна перевага графовик баз даних полягає в тому, що, оскільки рекомендація не базується на інформації споживачів, на неї не впливає проблема холодного запуску.

2.2 Кластеризація в теорії графів

Хоча єдиного визначення кластера в наборі даних графі немає, Шеффер [16] визначає кластер як “дані, такі, що елементи, присвоєні конкретному кластеру, подібні або пов’язані в певному, заздалегідь визначеному сенсі” [35]. Бар-Ілан та ін. визначили набір вимог, яким має відповідати набір даних, щоб бути прийнятим як кластер, визначений шляхами на графі [4]. У теорії графів шлях - це послідовність ребер, що починається з вузла N0 і закінчується у вузлі Nk. Визначення Бар-Ілана таке [4]:

- Кожен кластер повинен бути інтуїтивно підключений.
- Повинен бути принаймні один, а краще декілька шляхи, що з’єднують кожну пару вершин кластеру.
- Шляхи повинні бути підключені всередині кластеру.

- Два вузли в кластері повинні бути не тільки по шляху, який проходить через них, але і по шляху, який відвідує лише вузли в кластері.

Дж. М. Клейнберг визначає кластер з точки зору його щільності, відношення наявних ребер до максимальної кількості можливих ребер.Клейнбер вважає “хорошим” кластером, де підграф який утворює кластер, є щільним, але має відносно мало зв’язків з вузлами кластера до решти графі [30]. Це визначення спирається на досить туманну термінологію.

Проблема з цими теоретичними визначеннями кластерів полягає в тому, що вони непридатні до проблем реального світу. Однак існують алгоритмічні способи оцінки кластера. П’ять поширених підходів до виявлення кластерів. які обговорюються в книзі М. Нідхема та А.Е. Ходлера - це кількість трикутників і коефіцієнт щільності, міцно з’єднані компоненти, поширення етикетки та модульність Лувена [45]. Вони визначають вимірювання кількості трикутників, скільки вузлів утворює трикутники і ступінь до якої вузли мають тенденцію до групування. Сильно зв’язані компоненти як алгоритм, який знаходить групи, де кожен вузол доступний з кожного іншого взула в тій самій групі, дотримуючись напрямку зв’язку. Поширення міток як алгоритм, який визначає кластери, розповсюджуючи мітки на основі більшості околиць, і модульність Лувена як алгоритм, який максимізує передбачувану точність групування шляхом порівняння ваг і щільностей відносин із визначеною оцінкою або середнім значенням. Потім у книзі показано, що алгоритм сильно зв’язаних компонентів ідеально підходить для рекомендаційних систем. Однак визначення алгоритму надзвичайно високого рівня і немає експериментальних доказів, які б підтвердили це твердження.

Кількість трикутників і коефіцієнт щільності

Т. Шанканд Д. Вагнер провів ретельний аналіз кількості трикутників і коефіцієнта щільності у своїй статті 2005 року “Finding, Counting and Listing all Triangles in Large Graphs, An Experimental Study”. Вони визначили трикутник як тривузловий підграф і дослідили кілька методів підрахункі трикутників. Успіх алгоритмів підрахунку трикутників вимірювався часом

виконання та кількістю операцій з трикутником. Визначення операцій трикутника варіювалося між алгоритмами, але по суті було асимптотичним часом виконання.

Алгоритми були запущені в мережі з не орієнтованими графами, мережею доріг Німеччини та орієнтованим графом, базою даних фільмів IMDB. Для мереж біт метод ітерації вузла був найшвидшим у виконанні, він був найбільш асимптотично інтенсивним. Цей метод покладався на ітерацію кожного вузла та підрахунок навколишніх ребер для підрахунку кожного трикутника. Алгоритм із прямим хешуванням був найгіршим. Натомість цей алгоритм працював шляхом ітерації динамічних даних.

Міцно пов’язані компоненти

Алгоритм сильно пов’язаних компонентів був одним із перших алгоритмів виявлення скупчення. Цей алгоритм був винайдений Робертом Тарьяном у 1970 році. Алгоритм працює шляхом пошуку наборів вузлів, де всі вузли можуть бути досягнуті всіма іншими вузлами в обох напрямках, але не обов’язково безпосередньо. Еско Нуутіла та Ельяс Сойсалон-Сойнінен покращили цей алгоритм у 1990 році, що робить його здатним обробляти розріджені графи та тривіально компоненти [5].

Модульність Лувена

Модульність Лувена - алгоритм кластерицї графів, розроблений у 2008 році Вінсентом. Д. Бонделем та співавторами на основі розробки модульності в системі [5]. Модульність - це метод кількісного визначення міцності кластера в системі графів. Модульність - це значення від -1 до 1, яке вимірює щільність ребер всередині кластера графів порівняно з щільністю ребер за межами цього кластера. Математичне визначення модульності показано нижче.

$$Q = \frac{1}{2m}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \qquad (2.1)$$

Де A_{ij} - вага між вузлами і та j, k_i та k_j - це сума ваг вузлів, приєднаних до і та j відповідно. 2m - це сума всіх ваг ребер у графі, c_i та c_j - спільноти вузла, а δ - проста дельта функція [13].

Модульність була важливим компонентом у багатьох областях, які можна представити в мережі графів. Це включає всесвітню павутину, метаболічні мережі, соціальні моделі, і тому алгоритм кластеризації, заснований на модульності, був важливою віхою для досягнення [13].

Алгоритм модульності Лувена є двоетапним ітераційним процесом, який описано в рівнянні нижче.

$$\Delta Q = \left[\frac{\sum_{in} + 2k_{i,in}}{2m} - \frac{\sum_{tot} + k_i^2}{2m}\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

(2.2)

Де $\sum_{in}$ - це сума всіх ваг зв'язків всередині спільноти, у яку вона переходить, $\sum_{tot}$ - це сума всіх ваг посилань на вузли спільноти і.

З точки зору рекомендаційних систем, модульність Лувена була використана для створення систем рекомендацій фільмів із соціальних даних. У 2015 році Діпіка Лалвані та інші опублікували роботу, в якій використовували модульність Лувена для пошуку кластерів у графовій базі даних соціальних мереж для пошуку рекомендацій для фільмів. В іншому дослідженні, проведеному в 2013 році, використали дані IMDB для побудови графічної бази даних, яка описувала загальні фільми між людьми [32]. Хоча обидва експерименти дали точні рекомендації з кластера, обидва залежать від інформації користувачів. Поки що не проводилось розслідування щодо використання модульності Лувена в базі даних графів, яка містить лише інформацію про фільми.

Поширення Міток

Поширення міток- це система, яка працює шляхом позначення та перемаркування вузлів у графі до тих пір, поки не буде створено задовільний кластер міток. Ця техніка знайшла застосування в медичній сфері. Рольф А. Хекерманн та співавтори використовували поширення міток на МРТ-сканування мозку, щоб досягти більшої автоматичної анатомічної МРТ сегментації мозку, поєднуючи поширення мітки та злиття рішень з позитивними результатами.

У рамках математики були розроблені концепції для розв'язування складних задач неоднорідного бігармонійного рівняння з граничними умовами Діріхле. Це обговорювалося в статті Джіндонг Ванг "Лінійне поширення сусідства та його застосування". Однак реальне застосування цього алгоритму до складних математичних задач ще не повністю реалізовано.

Зх Ву та співавтори довели використання поширення міток у соціальних мережах для пошуку спільнот, що перекриваються, у свіїй роботі 2012 року "Збалансоване поширення кількох міток для виявлення перекриваючих спільнот у соціальних мережах". Поширення міток також виявилося корисним для стиснення графів, як було показано у 2011 році Паоло Болді та інших у своїй статті "Розповсюдження міток по шарах".

Однак з точки зору рекомендаційних систем, Хоу Кіанг та інші довели потенціал його використання, поєднавши його зі спільною фільтрацією. Це обговорювалося в їхній статті 2012 року "Метод персоналізованої рекомендації, заснованої на розповсюдженні кількох міток для виявлення спільнот, що перекриваються". У цій статті було досліджено набір даних MovieLens і виявлено ледь покращені результати щодо базового порівняння спільної фільтрації. У межах музични рекомендацій Бо Шао та інші використовували поширення міток для розробки системи рекомендацій, заснованої на графовій базі даних, що описує моделі доступу користувачів до музичної мережі NewWisdom,а також акустичні особливості пісень, які

слухав користувач. Це було написано в їхній статті 2009 року “Рекомендації щодо музики на основі акустичних характеристик та шаблонів доступу користувачів”.

Ще не проведено дослідження рекомендаційних систем із використанням кластеризації в графовій базі даних фільмів яка містить лише інформацію про фільм.

2.3 Проектування графової бази даних

Основними цілями цього проекту були, побудова графової бази даних з описом фільмів, аналіз методів кластеризації графів у застосуванні до зазначеної бази даних та дослідження методів кластеризації як методу рекомендації фільмів людям, на основі їхніх уподобань. Таким чином методологія була розділена на три розділи, створюючи та досліджуючи різні бази даних, виконання методів кластеризації в базах даних і використання знайдених кластерів для створення рекомендаційної системи.

Було протестовано різні види графових баз даних, щоб визначити оптимальний граф. Хоча граф може бути побудований правильно, з чіткими вузлами та зв’язками, функціональність графу можна дослідити лише шляхом аналізу результатів алгоритму кластеризації. Так само оптимальний метод кластеризації можна знайти лише шляхом порівняння результатів рекомендаційної системи.

2.3.1 Проектування графової бази даних

Було досліджено два підходи до побудови бази даних графів. Перший передбачав створення вузлів для кожного фільму та кожної властивості. Тобто для кожного фільму є вузол із міткою “ФІЛЬМ”, а для кожного актора, жанру, тощо в базі є вузол “АКТОР”, “ЖАНР”...Крї між цими вузлами будуть спрямовані відношеннями від вузла властивостей до вузла фільму. Основа цього графа полягає в тому, що вузли властивостей будуть групуватися з вузлами фільмів, після чого фільми можуть бути ідентифіковані з цих кластерів під час рекомендації.

Другий підхід полягав у побудові графа, який містив лише вузли фільму. Вузли фільму потім можуть бути з’єднані двонаправлено з ребрами, які представляють спільні властивості. Наприклад, якщо у фільмі є один із тих самих акторів, буде створено двонаправлений зв’язок із позначкою “СПІЛЬНИЙ_АКТОР”. Те ж саме було зроблено для інших розглянутих об’єктів. Очікувалося, що за допомогою графіків, розроблених за цим методом, кластери будуть формуватися на основі зв’язків між кожним фільмом, а не на основі вузлів властивостей, спільних між фільмами. Оскільки кластери будуть повністю складатись з фільмів, весь кластер можна використати для рекомендацій.

2.3.2 Використані дані

Даними, використаними для проектування графової бази даних, була база даних TMDb 5000 Movie. TMDb (The Movie Database) - це онлайн база даних про телебачення та фільми, створена спільнотою. Веб сайт збирає дані з 2008 року. Користувачі сайту збирають інформацію про 474 652 фільми, що поєднанні з публічним доступом, робить його чудовим джерелом для аналізу властивостей фільмів. Хоча публічно розміщені ата модеровані дані, як правило, не такі найдіні, як опубліковані дані. Обсяг веб-сайту передбачає, що їх можна використовувати для аналізу. База даних фільмів TMDb 5000 - це вибірка з 4083 фільмів із добіркою інформації про фільми.

Дані надійшли у вигляді двох файлів .csv, tmdb_5000_movies.csv і tmdb_5000_credits.csv. Перший містить детальну інформацію про кожен фільм, включаючи акторів і членів знімальної групи, а другий дає більш глибокий опис акторів і знімальної групи.

tmdb_5000_movies.csv

Стовпці даних tmdb_5000_movies.csv містять детальну інформацію про бюджет (дол. США), жанри, домашню сторінку, ідентифікатор (унікальний для кожного фільму), ключові слова, мову оригіналу, оригінальну назву, огляд, популярність, виробничі компанії, країни виробництва, дату випуску (дд/мм/рррр), дохід (дол. США), час виконання (хвилини), розмовні мови,

статус (випущено/не оприлюднено), слоган, назва, середнє число голосів, кількість голосів.

Дані взяти з Kaggle, онлайн-спільноти дослідників даних що належить Google.

Безперервні дані непридатні для графової бази даних, оскільки їх е можна використовувати для створення окремих, зручних для використання вузлів. Таким чином, бюджет, підрахунок голосів, середня кількість голосів, час виконання, дохід, дата випуску та популярність були відкинуті. Огляди, домашню сторінка та тегліни не можуть бути придатні для використання у вузлах, оскільки вони відрізняються від кожного фільму і не додадуть нічого цінного до архітектури графу. Статус виходу фільму не вплине на те, що можу стосуватися уподобань людини, і тому це теж ігнорувалося. Поточна назва була використана замість оригінальну, щоб зберегти узгодженість з іншими даними фільму.

Цей CSV мав один рядок для кожного фільму. Жанр, виробничі компанії, країни виробництва та мови спілкування мають кілька атрибутів, тому вони були представлені у форматі JSON. Зразок збережених даних можна побачити в таблиці 2.1

Жанр	**id**	**Мова оригіналу**	**Назва**	...
[{“id”: 28, “name”: “Action”}, {“id”: 12, “name”: “Adventure”}, …]	19995	en	Avatar	… ...

Табл. 2.1. Приклад стовпців взятих з tmdb_5000_movies

tmdb_5000_credits.csv

tmdb_5000_credits.csv мав подібний макет, хоча лише з чотирма стовпцями, Ідентифікатор фільму, назва, акторський склад, група. Ідентифікатор та назва фільму відповідають їхнім еквівалентам у tmdb_5000_movies, тоді як Cast і Crew, де рядки JSON описують акторів або членів групи. Рядок Cast JSON

містить ідентифікатор трансляції, що ідентифікує учасника трансляції в списку трансляцій, ім’я персонажа, кредитний ідентифікатор, що ідентифікує їх у кредитах, їхня стать (числові: 1=жінка, 0=чоловік), їх ім’я та кредитний порядок (порядок 0 означає головну діючу роль). Приклад рядка показано нижче:

[

{ "cast id": 242,

"character": "Jake Sully",

"credit id": "5602a8a7c3a3685532001c9a", "gender": 0,

"id": 65731,

"name": "Sam Worthington", "order": 0}...]

Рядок екіпажу містить ідентифікатор кредиту, відділ, в якому працював член екіпажу, стать (0/1), унікальний ідентифікатор, посаду та ім’я. Приклад рядка показано нижче:

[

{"credit_id": "52fe48009251416c750aca23", "department": "Editing",

"gender": 0,

"id": 1721,

"job": "Editor",

"name": "Stephen E. Rivkin"}...

]

В обох цих рядках єдиною корисною інформацією є ім’я співробітника та його ідентифікатор, щоб відрізнити їх.

2.3.3 Використані технології

Для створення графу використовувалися дві технології Python і Neo4j. Python використовується для підготовки даних TMDb 5000 у формат, який можна було використовувати для перетворення шуканих даних у граф. Neo4j було використано для створення графу.

Python і pandas

У Python для керування файлами csv використовується бібліотека pandas. Ця бібілотека може імпортувати файли csv та зберегти їх як об'єкт фрейму даних. Це дозволяє легко маніпулювати даними, які розглядаються як стандартна таблиця. Бібліотека здатна читати та зберігати рядки JSON як об'єкти фрейму даних. Це дозволило отримати доступ до властивостей, описаних як рядки JSON. Бібліотеку також можна використовувати для об'єднання таблиць, що робить її корисною для роботи з даними які розділені на кілька файлів.

Neo4j

Neo4j - це найпоширеніша у світі система керування графовими базами даних. Система працює на основі теорії графів, створюючи вузли, які містять дані, з'єднані ребрами ,які представляють деяке з'єднання даних. Neo4j дозволяє вільно будувати графи, включаючи створення нових вузлів і ребер. Коли вузол створюється, йому потрібно призначити мітку. Це встановлює відмінність між вузлами та можливістю запитувати граф. Вузол з міткою n позначається як (n). Окремі вузли можна відрізними один від одного, призначивши їм властивості під час їх створення. Це дозволяє робити запити до конкретних вузлів на графі. Кількість властивостей на один вузол необмежена.

Властивості можуть бути цілими, плаваючими, рядковими, булевими або масивом будь якого з типів перерахованих вище. Синтаксис створення вузла з міткою "MOVIE" з властивостями "title: The Matrix" та "id: 234" показано нижче:

```
Create (m:MOVIE {title: The Matrix, id: 234})
```

У Neo4j ребра в графі називаються відносинами. Ці відносини можуть бути ненаправленими, однонаправленими. Двонаправлені відношення утворюються двома однонаправленими відношеннями, протилежними один від одного. Як і у випадку з вузлами, зв'язкам необхідно присвоїти мітку. Відношення r між двома вузлами n і m представлено в Neo4j як: (n) - [r] -> (m).

Стрілка не є обов'язкова і вказує напрямок зв'язку. Відносинам можна призначити вагові коефіцієнти подібно до властивостей вузла. Це використовується для побудови зважених графів, які надають значення конкретним відносинам. Коли ваги призначаються, вони повинні бути позначені для правильного запиту. Створення зв'язку під назвою "acted_in" між вузлом (n) і вузлом (m) з вагою wt=0,5 показано нижче:

```
MATCH (n), (m)
CREATE (n) - [:acted_in :{wt: 0.5}] - (m)
```

MATCH є важливою командою в Neo4j, оскільки саме так здійснюється доступ до вузлів.

Запити в Neo4j

Перевагою Neo4j є його здатність запитувати базу даних. Програма використовує мову запитів під назвою Cypher для дослідження властивостей графу. Її можна використовувати для пошуку конкретних даних, побудови конкретних підграфів і застосування таких операцій, як підрахунок чи середнє. Вона також дозволяє маніпулювати властивостями вузла за допомогою числових і рядкових операцій.

Запити в Cypher орієнтовані навколо двох команд: MATCH і RETURN. MATCH використовується для пошуку, RETURN для повернення результату.

Між цим до певних вузлів можна отримати доступ і маніпулювати ними за допомогою однієї з їхніх властивостей. Наприклад, знайти вузом MOVIE з назвою "The Matrix" можна таким чином:

```
MATCH (m:MOVIE title: "The Matrix") RETURN m
```

Всередині цього запиту можна виконувати логічні, рядкові та числові операції так само як SQL. Для більш просунутих статистичних і математичних операцій необхідні пакети в Neo4j.

Алгоритми в Neo4j

Існують пакети, які дозволяють проводити більш складний аналіз. Два пакети, використані в цьому проекті, були APOC (Awesome Procedures on Cypher) і Graph Algorithms. APOC містить багато корисних функцій і особливо корисний для виконання різних статистичних рівнянь. Graph Algorithms використовується для застосування алгоритмів специфічних для теорії графів. Вони включають централізованість, пошук шляху, передбачення зв'язків і виявлення спільноти. Самі ці вбудовані алгоритми використовувалися для дослідження спільнот у базі даних фільмів TMDb 5000.

2.3.4 Граф з багатьма мітками

Перший досліджений граф складався з вузлів з декількома мітками. Кожен фільм був позначений його назвою та ідентифікатором. Кожна властивість цього фільму також було створена як вузол із відповідною міткою. Наприклад, вузли GENRE і ACTOR.

Підготовка даних

Щоб звузити граф з імпортованих даних у Neo4j, він повинен бути в одному csv. Коли csv зчитується в таблиці, можна перебирати рядки, а значення використовувати в побудові вузлів, зв'язків і властивостей. Оскільки дані були у формі двох файлів csv, їх потрібно було об'єднати в один таким чином, щоб за допомогою ітерації можна було побудувати потрібний граф.

Для графа з кількома мітками таблиця повинна мати стовпець для кожного досліджуваного фільму та властивості. З даними в цьому форматі кожен стовпець можна вважати міткою вузла, кожен унікальний запис є вузлом і відносини можна побудувати з кожного рядка.

Щоб це сталося, для фільму повинно бути кілька записів, щоб відповідати кожній із властивостей. Для цього було створено фрейм даних

pandas для кожного файлу csv. Окремі властивості кожного рядка були вбудовані в новий фрейм даних, стовпець для значення та стовпець для ідентифікатора фільму. Потім кадри даних були зовнішньо об’єднані за допомогою ідентифікатора фільму. Останніми стовпцями були ідентифікатор фільму, назва, мова оригіналу, дата випуску, ім’я режисера, ідентифікатор режисера, жанр, ім’я актора, ідентифікатор актора.

Були створені різні таблиці з різною кількістю акторів, щоб можна було дослідити вплив кількості акторів. Було створено сім таблиць, що містять від одного до семи акторів на фільм.

Створення графу

Потім граф було побудовано за допомогою Neo4j. З попередньої таблиці потрібно було створити чотири типи на вузлі: (ACTOR), (MOVIE), (DIRECTOR), (GENRE). Вони містили відповідне ім’я та ідентифікатор. Було побудовано такі відносини:

- (:ACTOR)-[:ACTED_IN]->(:MOVIE)
- (:DIRECTOR)-[:DIRECTED]->(:MOVIE)
- (:MOVIE)-[:IS_GENRE]->(:GENRE)

Процес створення графа за допомогою цього програмного забезпечення можна розбити на кілька кроків. По-перше, було накладено обмеження на створення вузлів, щоб вузли ACTOR мали унікальний ідентифікатор актора, вузли MOVIE мали унікальний ідентифікатор фільму, вузли DIRECTOR мали унікальний ідентифікатор режисера і щоб назва жанрів була унікальною Обмеження були накладені на ідентифікатори, а не на імена, оскільки це відрізняє спільні імена. По-друге, рядки таблиці виконуються з новим вузлом ACTOR для кожного ідентифікатора актора, цьому вузлу було надано ім’я та інформацію про ідентифікатор. Потім цей вузол був об’днаний у граф. Те саме було запущено для кожного типу вузла.

Щоб створити зв’язки, рядки таблиці були запущені знову. Цього разу функція Neo4j MATCH була використана для пошуку вузла, який збігається з ідентифікатором актора та вузла, який відповідає ідентифікатору

фільму і створює відносини ACTED_IN між ними. Це повторювалося для DIRECTED та IS_GENRE.

За допомогою цього методу було створено кілька графів для різної кількості таблиць акторів та версії графі з жанрами та без. Це дозволило перевірити вплив кожного параметру.

Схема графа з кількома мітками, що представляє дані фільму, можна побачити нижче.

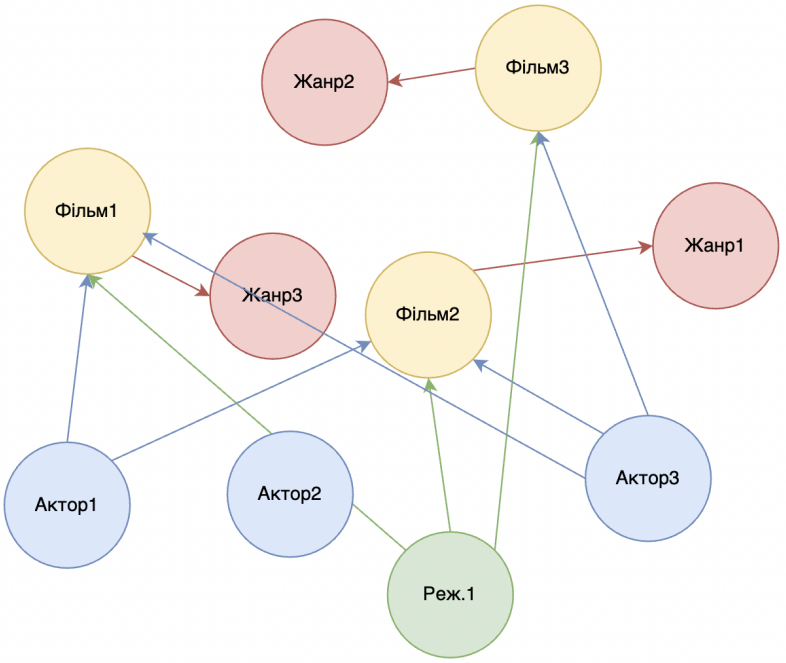

Рис. 2.2. Схема графа з кількома мітками

На зображенні показано різні мітки вузла на графу, червоний колір - жанр, сіний - актор, зелений - режисер, а жовтий - фільм. Зелені стрілки предаставляють зв’язок :DIRECTED, синій - :ACTED_IN, а червоні стрілки - відношення :HAS_GENRE.

2.3.5 Мітка одного вузла

Другий гарф, також побудований в Neo4j, складався тільки з вузлів позначених (MOVIE).

Деталі кожного фільму зберігалися як властивості його унікального вузла.

Дослідження властивості

Це метод дозволив включити більше властивостей, не боячись заповнення кластерів властивостями за рахунок фільмів. Властивости

включали в себе акторів, режисерів, жанри, ключові слова, продюсерські компанії, країну виробництва, мови оригіналу, та мови, якими розмовляють у фільмі. Для властивостей акторів було обрано перші п’ять перерахованих акторів. Це було зроблено тому. що актори меншого плану, мають багато другорядних ролей у фільмах, незалежно від стилю фільму. Передбачалося, що коли справа доходить до уподобань до кіно, часто помічають лише найпопулярніших акторів. Були використані всі режисери та жанри, що було досить специфічно. Ключові слова підкреслюють відмінні властивості і тому є більш корисними. Виробничі компанії, як правило, мають унікальні стилі, яким можна віддати перевагу, наприклад Disney.

Підготовка даних

Граф з однією міткою був побудований шляхом об’єднання двох оригінальних фільмів csv в один цілий файл. Усі дані JSON було перетворено в масив, що містить лише значення id.

Використовувалися лише значення id, оскільки це гарантує унікальність значень. Остаточний файл csv містив один рядок на фільм, а показники, які мають кілька значень, було представлення у вигляді масиву їхніх ідентифікаторів. У Neo4j csv був повторений і для кожного фільму був створення вузол. Під час створення вузлів кожному з них було надано властивість для кожної метрики з файлу csv. Оскільки властивості можуть бути числовими, булевими чи масивами, можна було легко описати всі жанри та акторів.

Створення графа

Коли csv було імпортовано в Neo4j, вузли були створені простою командою CREATE, використовуючи значення рядка для властивостей вузла.

Коли csv зчитується в Neo4j, кожен запис зчитується як рядок. Отже, неможливо просто прочитати масив як властивість вузла. Щоб протидіяти цьому. кожне значення масиву в csv було перетворено в рядок розділений комами. Під час зчитування значення в Neo4j команда “split”

використовувалася для поділу значень на масив Neo4j. Потім цей масив був призначений як властивість.

Відносини створювалися між вузлами за допомогою мови запитів Cypher. Cypher використовувався для порівняння вузлів. які мають однакове значення властивості, це було зроблено для властивостей, які не є масивами, таких як властивості вихідної мови. Для зв'зків між властивостями масиву відповідності були створені на основі перетину властивостей, які не є порожніми. Остаточний граф був містив вузли для кожного фільму та зв'ящки між тими, які мають подібні властивості. Запит можна побачити нижче:

```
MATCH (n:MOVIE), (n2: MOVIE)
WITH n, n2, apoc.coll.intersection(n.actor, n2.actor) AS common WHERE NOT common = []
AND NOT n = n2
CREATE (n) - [:shared_actor]->(n2)
```

Схема графу наведена на рисунку 2.3.

На цьому рисунку показано різні вузли фільму та те, як вони з'єднуються один з один. Червоні стрілки представляють спільний жанр, зелені стрілки представляють спільний фільм, а сині стрілки - спільного режисера.

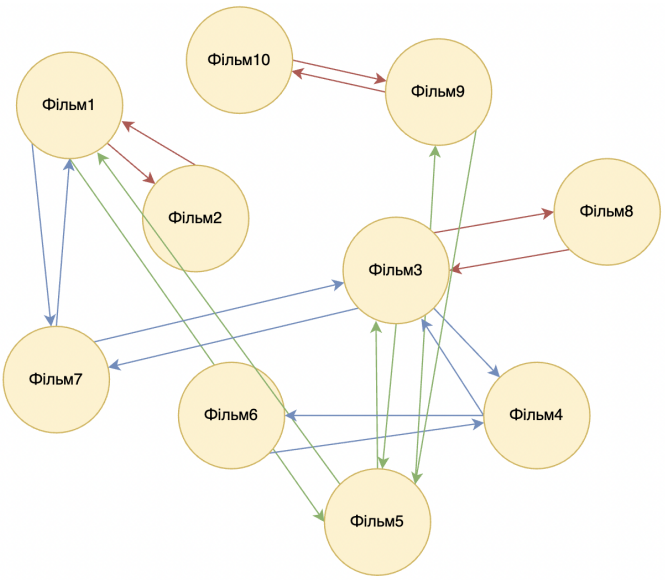

Рис. 2.3. Схема графа з однією міткою.

Додавання ваг до графа

Іншою властивістю графових баз даних, яка може вплинути на поведінку, є зважені відносини. Додавання ваг до графа можу допомогти визначити важливість відносин. Це може перешкодити тому, щоб щільність графа стала проблемною. Іншою причиною для дослідження ваг є те, що встановлення важливості між стосунками може ще більше ідентифікувати спільноти. Замість того, щоб усі відносини розглядалися однаково, акцент зосереджений на вузлах із сильнішими зв'язками, які можуть покращити результати алгоритмів кластеризації. Це особливо корисно для поширення етикетки та модульності Лувена.

Зважування для цього графа створено шляхом знаходження відношення кількості значень у властивості вихідного вузла до кількості значень на перетині вихідних і цільових вузлів. Це повертає значення від нуля до одиниці для всіх відносин. Це дає можливість розрізняти важливість відносин, надаючи всім типам відносин порівняльний діапазон. Важливо мати подібний діапазон значень для кожного типу зв'язку, щоб жодному зв'язку не надавалося більшого пріоритету. Це було спеціально зроблено в Neo4j шляхом зміни запиту Cypher щодо створення відносин у попередньому коді. Цей процес показаний нижче:

```
MATCH (n: MOVIE), (n2: MOVIE)
WITH n, n2 apoc.coll.intersection(n.actor, n2.actor) AS common
WHERE NOT common = []
AND NOT n = n2
CREATE (n) -[:shared_actor{weight: length(n.actor)/length(n.common)->(n2)}->(n2)
```

Незважаючи на те, що програмі було виділено максимальну пам'ять, завдання присвоювати зв'язки на основі країни виробництва була надто великою.

Закріплення ваги

Одна з проблем, яка може виникнути при зважуванні взаємозв’язків у цьому, полягає в тому, що кількість окремих ключових слів значно перевищує кількість жанрів у фільму, це в поєднанні з багатьма часто повторюваними ключовими словами означало, що перетин був набагато більше, залишаючи набагато меншу вагу для ключових слів. Також важко встановити важливість окремих властивостей у контексті даних.

У цих вузлах є багато значень, які дуже часто повторюються. Наприклад, ключове слово актор, що зустрічається в X кількості фільмів. Це не корисно, коли справа доходить до рекомендацій, оскільки воно не зосереджується на відмінних властивостях фільмів, які віддають перевагу людям. Набагато корисніше робити більше наголосу на менш частих словах.

Щоб протистояти цьому, дані містили частоту кожного елементу. Обернена частота кожного елемента була призначена як вага цього елемента. Призначення ваги кожному елементу дає кращу вказівку на дійсно важливі значення. Що стосується ключових слів, то чим рідше вживається слово, тим вище вага елемента.

Перетворення кінцевої ваги зв’язку на суму ваг елементів у перетині властивостей дає кращу індикацію впливу різних елементів властивостей га граф.

Оскільки відносини розвивалися таким чином, не було необхідності обмежувати кількість акторів. Оскільки цей підніх зважування більше зосереджується на унікальних елементах.

Щоб зробити це практично в Neo4j, властивості вузла повинні містити інформацію як про властивість, так і про їх частоту. Також мав бути спосіб зв’язати ці два. Нажаль, у Neo4j можуть бути лише базовими об’єктами, а це означає, що масив масивів не можна використовувати для порівняння індексів. Також не було способу пов’язати кожну властивість з окремою таблицею з ідентифікатором, оскільки могла зберігатись лише інформація про граф.

Це було вирішено шляхом зміни масивів у стовпцях властивостей. Масиви, які містили властивості, були змінені з [“id1”, “id2”, “id3”, … “idn”] на [“id1|frequency1”, “id2|frequency2”, ‘id3|frequency3”, … “idn|frequency”]. Під час побудови зв’язків між вузлами, після того як перетин було знайдено, значення в масиві властивостей можна було б розділити на “|” і індекс 1 містить частоту для кожного значення. Кожен елемент масиву може оброблятися за допомогою процесу, який називається розглядом списку. Цей процес створює новий список із визначеної функції. У цьому випадку створений список є списком частот масиву перетину. Код для цього можна побачити нижче:

```
WITH [x IN common | toFloat(split(x, ‘|’)[1]] AS result
```

Тут кожен елемент у масиві перетину розбивається, перетворюється на float, а потім вибирається другий елемент який містить частоту. Потім цей масив можна підсумувати за допомогою функції apoc.coll.sum() і встановити як вагу зв’язку. Однак у Cypher, під час використання списку, посилання на початково відповідні вузли (n), (n2) втрачаються. Отже, під час спроби об’єднати функції в запиті, як показано нижче:

```
MATCH (n:MOVIE), (n2:MOVIE)
WITH n, n2, apoc.coll.intersection(n.keywords, n2.keywords) AS common
WHERE NOT common = []
AND NOT n = n2
WITH [n IN common | toFloat(split(x, ‘|’)[1]] AS result
CREATE (n)-[:shared_actor{wt: apoc.coll.sum(result)}]->(n2)
```

Результатом є створення двох нових пустих вузлів, з’єднаних зв’язком “shared_actor” для кожної комбінації (n), (n2).

Щоб подолати це, запит на створення перетину був замінений, щоб повернути n.title, n2.title, common_keyword, common_actor … Це призводить до таблиці з усіма комбінаціями які дозволені запитом. Потім цю таблицю було збережено як файл csv.

Потім був створений порожній екземпляр Neoj4. У цьому випадку csv завантажувався рядок за рядком, як зазвичай. Цього разу значення рядка для common пройшли через розуміння списку та підсумовування. Потім суму було повернуто й збережено у файлі csv. Цей файл csv містив ваги для кожного типу зв'язку для кожного комбінації у тому ж порядку, що й попередній csv. Ці файли csv були об'жнані за допомогою pandas. Після цього в іншому порожньому екземплярі Neo4j було поміщено обмеження, щоб зробити назву кожного вузла унікальною. Після цього остаточний csv читався рядок за рядком, об'днуючи вузли та зв'язки.

Поєднання відносин

Ще одним кроком у боротьбі з щільністю графіка було об'єднання всіх зв'язків між вузлами в одне єдине відношення. Зменшення кількості зв'язків допомагає збільшити щільність гарфу, а спрощення міток зв'язків може підвищити продуктивність алгоритмів кластеризації. Окрім зменшення щільності графа, об'єднання зв'язків також може зробити зрозумілішим, як кожен вузол пов'язаний. Одна мітка відносин також може покращити якість виявлення спільноти.

Можна побудувати запит, який використовує функції агрегації для визначення сумарної ваги всіх зв'язків між двома вузлами. Тут для кожного вузла були агреговані вагові коефіцієнти існуючих зв'язків, які використовувалися як вага для ного зв'язку між вузлами. Інші відносини між вузлами були видалені.

Існують три основні методи агрегування, які можна використовувати для ефективного об'єднання зв'язків, зосереджуючись на числових вагах. Це Count, Average і Sum. Count повертає кількість зв'язків між вузлами. Це хороший варіант для створення ваги з незважених стосунків. Замість того, щоб алгоритм інтерпретував декілька зв'язків, підрахунок можна використовувати для надання чіткого співвідношення, яке чітко підкреслює силу зв'язку.

Однак це не вигідно для агрегування зв'язків із ваговими показниками, оскільки воно ігнорує важливість існуючих зв'язків, надаючи кожному з них однакове значення. Існує три способи усереднення ваг, середнє, медіана та мода. Медіана не була корисною для цього випадку, оскільки значення float об’єднуються, а це означає, що загальних ваг буде дуже мало. Використання медіани ігнорує вплив найсильніших стосунків. Середнє може бути корисним способом агрегування, однак воно ігнорує кількість зв’язків між двома вузлами. Два вузли з вагами відношень [0,5, 0,3, 0,4] мають те саме середнє, що й два вузли з вагами відношень [0,5, 0,3].

Незважаючи на те, що Neo4j має вбудовану функцію агрегації, ці функції агрегують ваги всіх зв’язків на графі. Неможливо використовувати їх для агрегування лише відносин у певних вузлах. Для цього необхідно зіставити та призначити змінну всім значенням. Базова відповідність тут не працюватиме, оскільки цей запит відкидає нульові значення. Це призводить лише до вагових зв’язків для зв’язків, які існують між усіма вузлами.

```
OPTIONAL match (n)-[s:shared_keywords]->(m)
OPTIONAL match (n)-[s:shared_production_country]->(m)
OPTIONAL match (n)-[s:shared_director]->(m)
OPTIONAL match (n)-[s:shared_actor]->(m)
```

Так збираються всі зазначені відносини між n і m як набір об’єктів. Список, що містить заголовки для кожного (n) і (m), був об’єднаний у список, що містить інший список з усіма ваговими значеннями [n.title, m.title, [r.wt, s.wt, t.wt, u .wt]]. Останній елемент підсумовується за допомогою функції apoc.coll.sum(). Однак, оскільки було використано необов’язкову відповідність, цей список містив нульові значення, які викликають помилку під час запуску. Для протидії цьому була використана функція фільтра Neo4j. Ця функція перевіряла кожен елемент на IS NULL.

Потім це значення було замінено на нуль. Повна функція, показана нижче:

```
WITH [n.title m.title, apoc.coll.sum(FILTER(x IN [s.wt, r.wt, u.wt, t.wt] WHERE x IS NOT NULL ))] AS combination.
```

Список n.title, m.title та їх сукупна вага зв'язку було присвоєно змінній комбінації, щоб мати доступ до неї під час створення нового зв'язку.

У тому ж запиті було використано інше речення MATCH. Конкретні вузли були ідентифіковані шляхом індексації списку комбінацій. Нове співвідношення було створено з вагою, знайденим комбінацією індексації. Ця частина запиту показана нижче:

MATCH (n: MOVIE title: combination[0])-[w]->(m:MOVIE title: combination[1])
MERGE (n)-[:combinewt:combination[2]]->(m).

Тут w є лише місцем для відносин. MERGE використовувався замість create, щоб уникнути дублювання відносин. Після цього вихідні зв’язки були видалені, залишивши лише комбіновані вагові зв’язки.

2.4 Кластеризація

Дослідження кластеризації було другою основною метою проекту. Метою тут було визначити, чи може граф бази даних фільмів виконати успішну кластеризацію з метою рекомендації фільмів. Було два ключових моменти аналізу; знайти оптимальний граф і поєднати його з оптимальним алгоритмом. Якість графу та якість алгоритму були перевірені одночасно із застосуванням алгоритму. Якщо кластери були успішно виявлені, то це дозволило б довести, що можна побудувати граф з метою кластеризації і можна використовувати алгоритми для виявлення спільнот. Створені тут кластери можна було б потім використати для створення рекомендаційної системи

2.4.1 Показники порівняння кластерів

Оскільки числові методи визначення зв’язків непридатні, метриками, за якими оцінювалися алгоритми, були розмір кожного кластера та кількість вузлів у кожному кластері. Це є найкращим показником для використання рекомендаційних систем, оскільки кожен фільм повинен бути в межах кластера, а кожен кластер повинен містити відповідну кількість кластерів для створення рекомендацій. Ідеальна система рекомендацій забезпечила б метод рекомендації кожного фільму. Це вимагало б мінімізації кількості кластерів з

одним вузлом і кластерів, що містять лише один вузол фільму. Якщо граф групується таким чином, що більшість фільмів утворює один великий кластер, метод не дасть достатньо вузької рекомендації. Ці показники використовувалися для порівняння різних алгоритмів і різних структур графів під час спроби визначити найкращу структуру графу та найкращий алгоритм для виконання рекомендаційного тестування.

2.4.2 Алгоритми кластеризації Neo4j

Ця кластеризація була здійснена за допомогою пакета Neo4j Graph Algorithms.

Алгоритм Лувена

Реалізація алгоритму Лувена в Neo4j показана нижче:

CALL algo.louvain.stream(label:String, relationship:String, weightProperty: ‘weight’, defaultValue: 1.0)

YIELD nodeId, community

У функції algo.louvain.stream є п'ять аргументів, мітка, зв'язок, weightProperty, значення за замовчуванням і паралельність. Перші два аргументи визначають, який вузол мітки та зв’язок потрібно аналізувати. Щоб використовувати всі мітки та зв’язки, це поле залишається порожнім. Властивість weight і значення за замовчуванням дають можливість включати ваги в алгоритм. weightProperty можна використовувати для визначення ваги зі змінної weight. Yield повертає результати з алгоритму Neo4j Graph. "nodeId" - це ідентифікаційний номер вузла. Його можна використовувати для визначення фільму та його властивостей. "community" - це ціла мітка спільноти.

Вагові коефіцієнти в цих системах відрізнялися в залежності від відносин. Щоб включити це в інформацію, потрібно було вказати вагу. Це було зроблено шляхом створення пропозиції відповідності, яка повертає змінну зв’язку в аргументі мітки вузла алгоритму. Аргумент зв’язку потім визначається іншою функцією відповідності. Цього разу за допомогою SOURCE та TARGET. Використовуючи раніше відповідні вузли як джерело

та ціль оператора, вагу зв’язку можна повернути як “weight”. Потім це може бути підібрано за допомогою алгоритму. Це показано нижче:

CALL algo.louvain.stream((‘MATCH (n) RETURN id(n) as id’, ‘MATCH (n1) -[r:combine]->(n2)

RETURN id(n1) as source, id(n2) as target, r.wt as wight’, graph: ‘CYPHER’, write:true))

Визначення змінної "graph" робить це проекцією Cypher. Це означає, що кластеризація будується на основі проектування підграфів, запитаних у аргументах мітки та зв’язку.

Ця зміна алгоритму є тим самим методом для включення ваги в усі алгоритми Neo4j 'Graph Algorithm'

Поширення міток

Алгоритм поширення мітки Neo4j, який використовувався, показаний нижче:

CALL algo.labelPropagation.stream(label: String, relationship: String, iterations:1, weight Property: ‘weight’, writeProperty:’partition, direction: ‘OUTGOING’)

YIELD nodeId, label

Алгоритм працює так само, як і алгоритм Лувена. Однак є три нових аргументи; "iterations", "writeProperty" і "direction". "iterations" встановлює максимальну кількість ітерацій, через які буде виконуватися алгоритм. Щоб підтримувати досить малий час виконання, для кожного запуску було вибрано значення 5. "writeProperty" - це властивість, до якої записується алгоритм. Як і у випадку з Лувеном, для визначення ваг знадобилося початкове відповідність. Це було зроблено таким же чином.

Підключені компоненти Neo4j

Функція Neo4j для виконання підключеного компонента показана нижче:

CALL algo.unionFind.stream(label: String, relationship:String, weightProperty:’weight’, threshold:0.42, defaultValue: 1.04)

YEILD nodeId, setId

Цей алгоритм працює майже так само, як і раніше. Єдина відмінність полягає у включенні «treshold», який є плаваючим значенням, що визначає поріг ваги відносин. Нижче цей зв’язок відкидається. "setId" надає значення з плаваючою чисельністю, що визначають кластер, як і раніше.

Компоненти Neo4j, які сильно пов’язані

Алгоритм струнно зв’язаних компонентів набагато простіше в порівнянні, як можна побачити в синтаксисі нижче:

CALL algo.scc.steam(label: String, relationship: String)

YEILD nodeId, partition

Немає методу включення ваг в цей алгоритм. Єдині аргументи, які можна змінити, це мітки вузлів і досліджувані зв’язки. Тут були використані всі мітки та зв'язки. Вихід "partition" - це ціле число, що відповідає кластеру, як і в інших алгоритмах.

2.4.3 Графи з кількома мітками

Вибір алгоритму

Кожен граф містив різну комбінацію функцій фільму, тому алгоритми працюватимуть по-різному. Однак єдина спільна риса цих графів — відсутність трикутників. Єдині відносини, які можуть сформуватись із цих даних, — це мононаправлені від вузла, що не є фільмом, до вузла фільму. Отже, немає можливості трьох ребер у трьох вузлах. Підрахунок трикутників і функція щільності покладаються на формування трикутників у базі даних [], що робить ці методи непридатними для використання в цьому контексті.

Отже, методами, які були досліджені, були поширення мітки та модульність Лувена.

Застосування алгоритмів

Алгоритми запускаються з використанням вбудованих функцій Neo4j. Це дозволило застосовувати його безпосередньо до створених графів. Перші досліджені графи складалися з вузлів :MOVIE та вузлів :ACTOR із відношенням :ACTED_IN. Були досліджені графи між одним актором на

фільм і сімома акторами на фільм. Кількість акторів була обмежена через інтенсивність обчислень, створюючи більше зв'язків і вузлів.

Після того, як результати були отримані для них, були додані вузли :GENRE і :HAS_GENRE. Були додані всі двадцять жанрів і надано повний список жанрів для кожного фільму. Після цього було додано мову оригіналу. Оскільки обидва досліджені алгоритми є недетермінованими, кожен алгоритм було п’ять разів запущено, а результати усереднювався.

2.4.4 Граф з однією міткою

Оскільки граф з однією міткою містить лише двонаправлені зв'язки, можна було дослідити всі чотири обговорюваних алгоритми, компоненти сильного зв'язку, кількість трикутників, модульність Лувена та поширення мітки. Однак алгоритм сильнозв’язних компонентів Neo4j немає аргументу ваги, що робить його непридатним для дослідження зважених графіків. Оскільки модульність Лувена та поширення мітки недетерміновані, їх довелося запускати кілька разів, щоб врахувати різні результати. Ці алгоритми виконувались п’ять разів на кожному графі. Інші методи були детермінованими, що означає, що результат буде однаковим під час кожного запуску. Отже, не потрібно було запускати графік кілька разів.

2.4.5 Порівняння результатів

Комбінація графу та алгоритму, яка дала найкращі результати, була запущена знову, цього разу були створені та зібрані таблиці, що визначають кластери. Якщо оптимальні результати були досягнуті за допомогою детермінованого підходу, то потрібно взяти лише один набір результатів. Якщо оптимальний метод був недетермінованим, то бралися п’ять наборів результатів. Це повинно було забезпечити надійність результату.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА АНАЛІЗ ВИКОРИСТАНИХ ПІДХОДІВ

Це дослідження має на меті вирішити три основні цілі: побудова бази даних графів з даних фільмів; досліджувати кластеризацію в базах даних фільмів і досліджувати, наскільки добре ці кластери можна використовувати для рекомендації фільмів. Аналіз був розбитий на три аспекти: дослідити, наскільки добре були побудовані графіки, дослідити, наскільки добре працювали алгоритми кластеризації, дослідити, як кластери працювали як метод рекомендації. Оскільки було досліджено дві структури графа, обидві були проаналізовані окремо. На одному позначеному графу було досліджено, наскільки добре працюють усі різні методи кластеризації, а на графі з кількома мітками було досліджено, як зміна включених вузлів вплинула на результати застосовних алгоритмів.

3.1 Одновузловий граф

3.1.1 Побудова графу

Першою частиною аналізу було дослідження того, як будувався графік. Це включало перегляд кількості вузлів і зв’язків. Початковою ознакою успішного графіка є наявність вузла, створеного для кожного фільму, що містить необхідні властивості. Інша річ, яку спочатку слід розглянути, це те, чи вдало з’єднуються стосунки та чи є значуща кількість зв’язків. Наприклад, не було б сенсу, щоб вузол з’єднувався з кожним іншим вузлом для будь-яких властивостей. Також важливо перевірити, чи існує розумна кількість зв’язків на вузол.

Граф з одним вузлом був побудований шляхом спочатку встановлення всіх вузлів в одному запиті, а потім запиту на створення кожного зв’язку. Кожне відношення було засноване на перетині властивостей вузла. Вони додають по одному запиту. Таблиця 3.1 нижче показує результати запитів у момент їх введення.

Відносини	Кількість	Кількість на вузол	Витрачений час (мс)
Спільні актори	165,496	34,4567	471,996
Спільні реж.	15,902	3,3108	159,199
Спільні ключові слова	847,246	207,505756	811,661
Спільні жанри	8,831,308	2162,94587	369139
Спільна компанія	536,698	111,742244	98354

Табл. 3.1. Кількість доданих зв'язків для кожної властивості

У комп'ютері не вистачило оперативної пам'яті коли порівнювались країни, хоча для комп'ютера вона була максимальною – 7 ГБ. Це тому, що майже все виробляється в США. При спробі побудувати рекомендаційні системи втрата інформації про фільм може спричинити негативний ефект. Однак у цьому випадку, включаючи країну походження, для рекомендаційної системи не є добре, оскільки такий загальний результат не може запропонувати відмінності між фільмами. Також побудова спільного зв'язку між кожним вузлом може викликати плутанину в алгоритмах графів.

Згідно з дослідженнями, проведеними веб-сайтом про фільми та освіту Stephen Follows [19], у середньому актор матиме п'ять найпопулярніших ролей у фільмах. Оскільки лише п'ять найпопулярніших акторів у списку були збережені в масиві, слід очікувати 25 відносин на вузол.

Кількість створених акторних стосунків становить 34,4567, що більше в 1,37 разів ніж очікувалося. Ця різниця є розумною на основі зроблених припущень. Більше значення також має сенс, оскільки більш популярні фільми мають менший вибір акторів. Зазвичай можна очікувати, що на один фільм буде від одного до трьох режисерів. Друга дослідницька робота Стівена

Фолусса свідчить, що середній режисер зніме лише два фільми. Отже, очікувана кількість спільних відносин режисера на вузол буде від двох до шести. Досягнуте тут значення 3,3108 добре вписується в цей прогноз. Визначення жанрів залежить від самого TMDb, тому важко порівняти ці цифри зі статистикою фільмів. Однак TMDb пропонує вибір лише з двадцяти жанрів. Ці результати свідчать про те, що 45% фільмів мають однакові жанри. Якщо порівняти кількість фільмів у даних, 4803, з кількістю жанрів, то зрозуміло, що буде велике перехрестя жанрів.

Ключові слова TMDb охоплюють ряд властивостей фільму. Сюди входять загальні описи, такі як “Action” або більш конкретні терміни, такі як «Джейсон Ворхіз», що посилаються на персонажа з серії «П’ятниця, 13-а». Таким чином, для кожного вузла має бути велика кількість спільних ключових слів. За даними Box Office Mojo, веб-сайту IMDb, який відстежує доходи від касових зборів, 80% частки кіноринку належить п’яти постановкам [46]. На цьому графіку міститься 111,74 зв’язків компанії спільного виробництва на вузол. Це 2,3% від загальної кількості вузлів. Однак слід враховувати, що багато продюсерських компаній публікують фільми під іменами активних компаній. Наприклад, Disney публікує «Зоряні війни» через LucasFilm [10]. Розглядаючи вищезазначене, можна припустити, що цей метод побудови графічної бази даних для зберігання інформації про фільм добре працює як для зберігання бази даних фільмів TMDb5000, так і для представлення загальної кіноіндустрії. Хоча для включення країни походження можна використовувати більш потужний комп’ютер.

3.1.2 Основне зважування

При побудові системи з ваговими коефіцієнтами кожне співвідношення додавалося окремо. Така ж проблема з процесорною потужністю виникла при розгляді країни-виробника. Однак у цьому випадку вплинуло й кількість жанрів. Ця додаткова втрата інформації не є ідеальною під час спроби представити дані фільму. Однак це принесло додаткову перевагу вагових

показників графіка. Втрата конкретних деталей про фільм призводить до кращого уявлення про те, наскільки міцно пов’язані вузли.

Актори, режисери та ключові слова, усі змогли налагодити стосунки. Кількість зв'язків показано нижче в таблиці 3.2

Відносини	**Кількість**	**Кількість на вузол**	**Витрачений час(мс)**
Спільні актори	165,496	34,4567	10,362
Спільні реж.	15,902	3,3108	4,262,39
Спільні ключові слова	847,246	207,505756	17,524,651
Спільна компанія	536,698	111,742244	194,304
Середнє число	387,399755	80,657	7126886

Табл. 3.2. Кількість доданих зв’язків для кожної властивості.

Отримана кількість зв’язків була такою ж, як і на попередньому графіку. Це було очікувано, оскільки єдиною бажаною зміною були додані ваги. Це показує, що доданий процес не вплинув на зміну структури графу. Однак час роботи цього зайняв у середньому в 11,6 разів більше часу. Зробивши його набагато більш складнішим у обчислювальних аспектах. Зразок графіка можна побачити нижче на рисунку 3.1

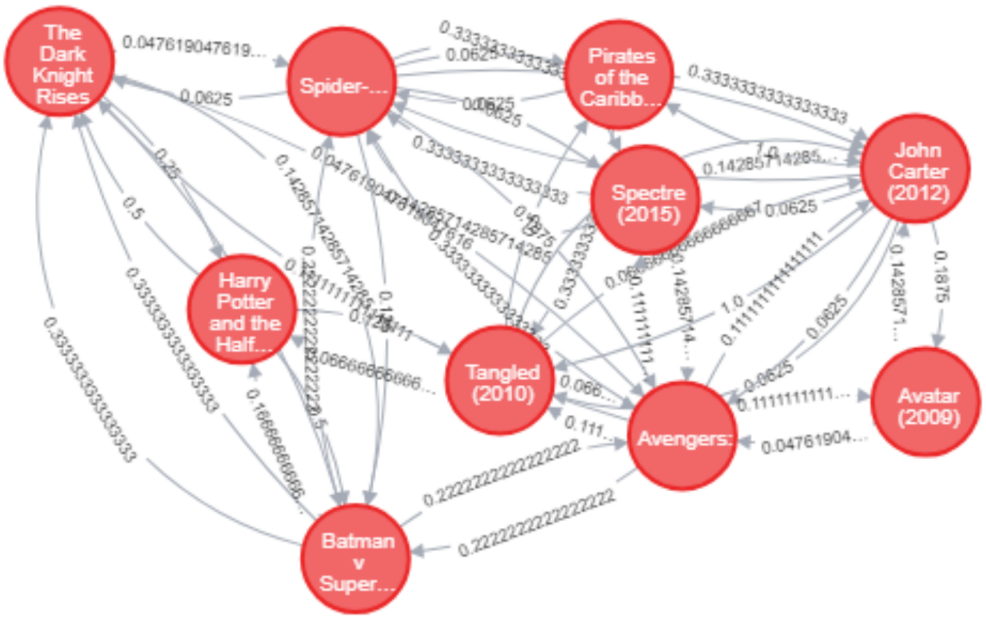

Рис. 3.1. Зразок графу з коефіцієнтом перетину як вагою

Цей метод був успішним у створенні бази даних графів, що з'єднує вузли фільмів із зваженими зв'язками, визначеними спільними властивостями фільму. На рисунку 3.2 наведено діаграму порівняння ваг різних відносин.

Середня вага актора становила 0,2048, що означає, що в середньому на перехресті був лише один актор, оскільки кількість акторів для кожного фільму була встановлена на п'ять. Розподіл навколо цього результату дуже малий, як видно з рисунка 3.2. Хоча є фільми, які поділяють набагато більше фільмів. Кілька сильніших результатів допомагають створити міцніші стосунки, які можна відрізнити від основної кількості акторів.

Дивлячись на рисунок 3.2, можна побачити, що середня вага відносин спільного режисера дорівнює 0,972 з дуже малим діапазоном. Це пов'язано з тим, що на фільм є лише один режисер, а це означає, що будь-який перетин, швидше за все, буде таким же, як об'єднання властивостей, що дасть співвідношення одиниці.

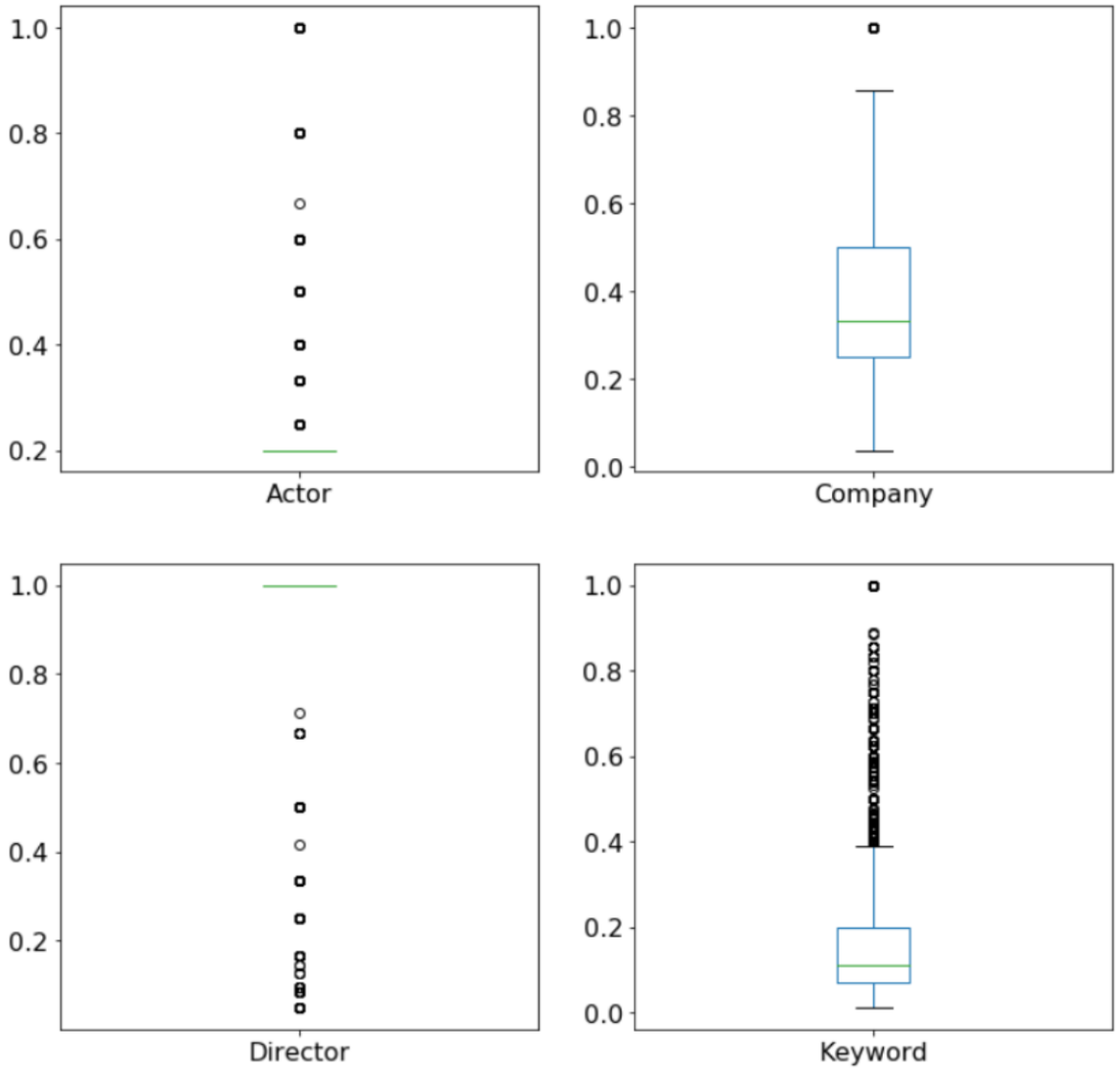

Рис. 3.2. Графіки ваг із базового зважування

Спільні ключові слова мають дуже мале середнє значення 0,165, хоча є деякі дуже сильні відхилення та більший розподіл, ніж інші. Компанія демонструє набагато більш рівномірний розподіл із середнім значенням 0,416.

Проблема з показаним контрастом у вагових показниках полягає в тому, що майже всі відносини з режисером переважають зв'язки з ключовими словами, що потенційно робить дані про ключові слова зайвими. Однак, оскільки кількість зв'язків спільного режисера на вузол набагато нижча, ніж кількість спільних зв'язків із ключовими словами на вузол, це означає, що зв'язки спільного режисера є більш корисним ідентифікатором зв'язків під час кластеризації.

3.1.3 Фіксовані зважування

При створенні графіка на основі частоти документа, властивості частоти були успішно створені в бажаному вигляді, як описано. Однак під час спроби налагодити стосунки виникли проблеми. Оскільки процес заснований на створенні таблиці, яка містить вузли для з'єднання та їх перетин. Через кількість можливих вузлів і з'єднань виготовлення цієї таблиці було занадто вимогливим з точки зору обчислень. Хоча були спроби з усіма дослідженими властивостями, жоден з спроб не зміг створити таблицю без збоїв.

Щоб дослідити доказ концепції, методику випробували з графом з двадцяти фільмів. Процес був таким же, але копія csv, що містить лише перші двадцять записів, була прочитана в Neo4j. Була створена невелика версія потрібного графу. Візуалізація цього графу, що відображає вагові коефіцієнти зв’язків shared_keywords та їх ваги, можна побачити на рисунку 3.3.

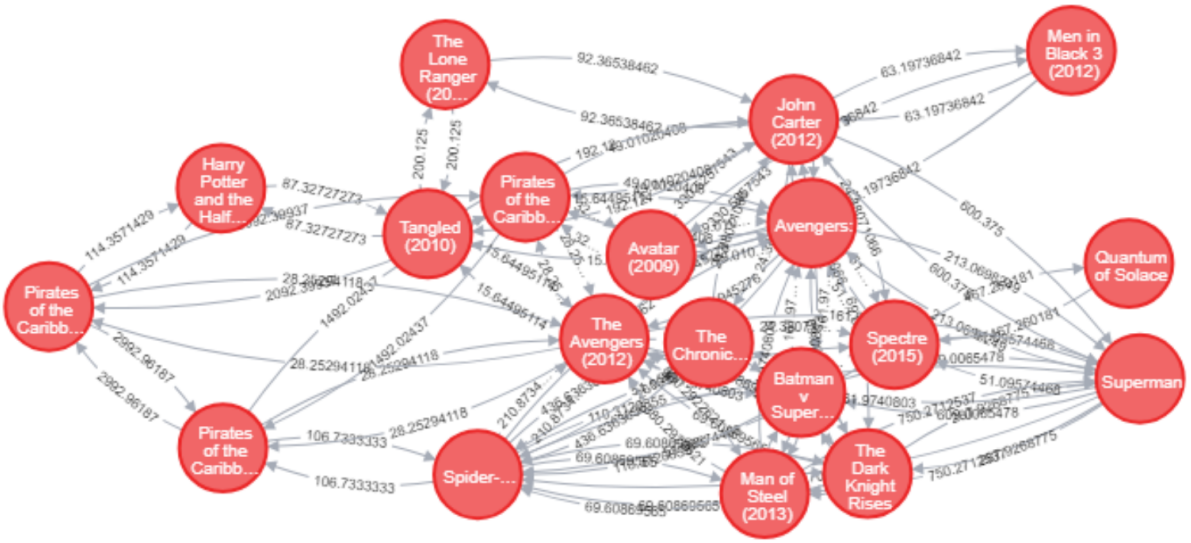

Рис. 3.3. Граф, що показує вагу частоти спільних ключових слів.

Хоча на жаль, графік не може бути досліджений далі, підтвердження концепції показує його здатність використовувати частоту документа як вагові коефіцієнти для відносин.

3.1.4 Поєднання відносин

Цей набір містить 847 246 властивостей, створив 847 246 зв’язків і був завершений через 5 439 016 мс. Код успішно встановив одну властивість для

кожного нового зв’язку. Це 176,4 відносини на вузол. в 2,18 рази більше середніх відносин на вузол від основного результату зважування. З огляду на те, що середні значення скошені екстремальними значеннями, це говорить про те, що всі відносини були успішно об’єднані.

Нижче на рисунку 3.4 наведено зразок бази даних комбінованого графіка зв’язків.

Рис. 3.4. Зразок агрегованого графу відносин

З цього зразка видно, що існує багато зв’язків на один вузол, однак є лише два зв’язки між будь-якими двома вузлами. На рисунку 3.5 можна побачити діаграму вагових значень агрегованих відносин.

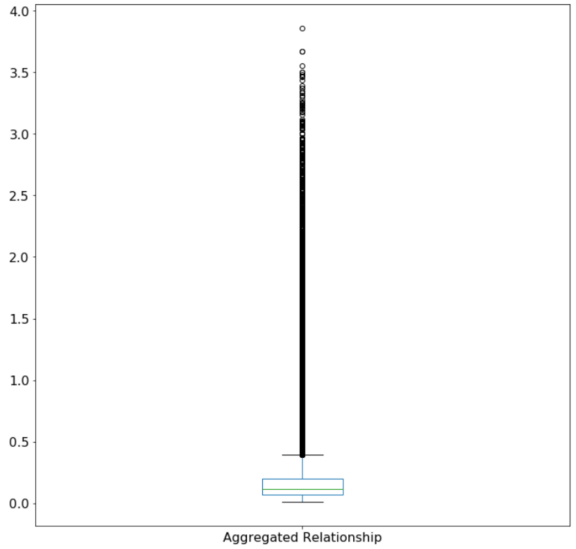

Рис. 3.5. Діаграма ваг агрегованого графу зв'язків.

Можна помітити, що і середнє значення, і інтерквартильний діапазон дуже низькі. Однак існують дуже великі аномальні значення. Це може свідчити про ефективний потенціал кластеризації, оскільки кілька неймовірно сильних ваг встановлять чіткі зв'язки між багатьма слабкими зв'язками.

3.1.5 Кластеризація

Чистий граф

Першим проаналізованим графом був чистий одновузловий граф без доданих вагових показників. Першою мірою було число трикутників і коефіцієнт кластеризації. Для відображення різних результатів було створено діаграму, як показано на рисунку 3.6.

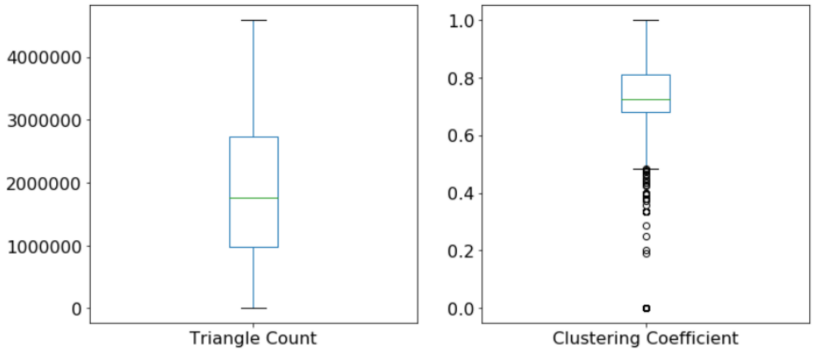

Рис. 3.6. Одновузловий граф.

Дивлячись на кількість трикутників, можна побачити, що існує величезний діапазон від нуля до майже 5000000. Це не дуже хороший знак, оскільки було б бажано, щоб усі вузли містили багато трикутників. Аналогічно, хоча середній коефіцієнт кластеризації 0,786 є скоріше високою, поширення результатів означає, що багато вузлів не знаходяться в межах відповідного кластера. Для системи рекомендацій важливо, щоб усі фільми були згруповані в чітко визначені кластери, щоб можна було дати ґрунтовні рекомендації. Другим алгоритмом, запущеним на цьому графіку, був алгоритм сильно зв'язаного компонента. На рисунку 3.7 нижче показано розміри кластерів, знайдених за допомогою алгоритму.

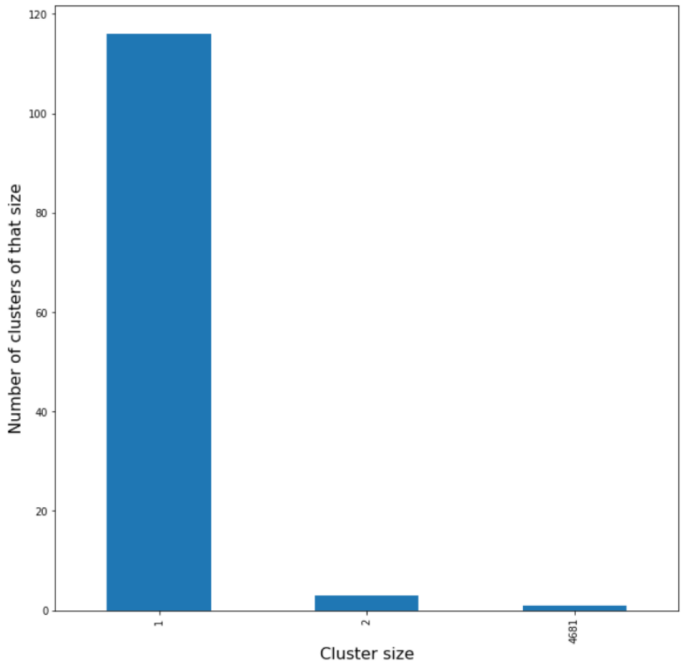

Рис. 3.7. Графік, що показує розміри різних кластерів, а також різну кількість кластерів цього розміру за допомогою алгоритму сильно зв'язаних компонентів.

Цей рисунок показує, що кластеризація розподілена дуже погано. Є один кластер, який містить 4681 із 4803 фільмів, решта вписується в кластери розміру 1 або 2.

Ймовірно, це пов'язано з щільністю графіка. Оскільки для кожного аспекту фільму існують взаємозв’язки, загальна кількість зв’язків становить 10396650. Це те, що створює таку велику кількість трикутників. Оскільки

трикутників утворено так багато, усі вузли занадто пов’язані. Виконання алгоритму підключеного компонента призвело до тих самих значень, що й для сильно пов’язаних компонентів. Як видно нижче на рисунку 3.8.

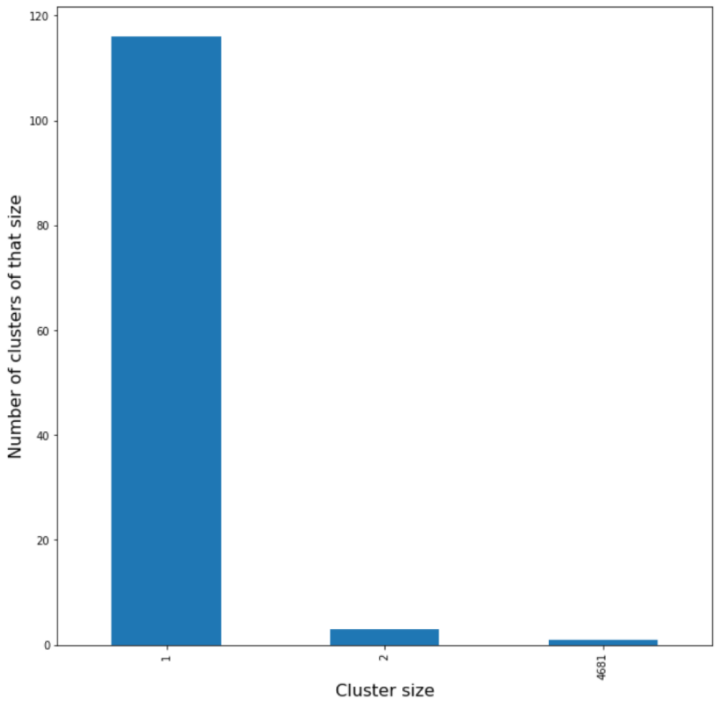

Рис. 3.8. Графік, що показує розміри різних кластерів, а також різну кількість кластерів цього розміру за допомогою алгоритму підключеного компонента.

Це пов’язано з єдиною відмінністю алгоритму в тому, що сильно зв’язані компоненти знаходять групи вузлів, які можна досягти, дотримуючись напряму зв’язку. Алгоритм зв'язаних компонентів, однак, ігнорує напрямок. Ця різниця не впливає на графіки цієї структури, оскільки кожен напрямний вузол між кожним відношенням має інше відношення в протилежному напрямку. Таким чином, вузли будуть доступні незалежно від напрямку зв’язків.

Після запуску алгоритму Лувена на цьому графіку результати були нанесені на стовпчасту діаграму, як показано нижче на малюнку 3.9.

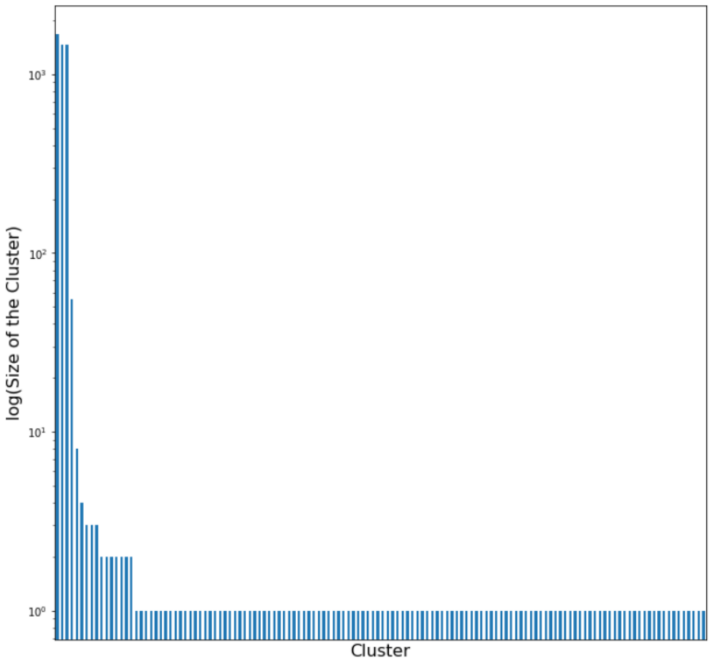

Рис. 3.9. Гістограма, що показує логарифм розміру кожного кластера, виявленого за допомогою алгоритму кластеризації Лувена

На цьому графіку кожен стовпчик представляє кластер, а вісь y — це логарифм 10 базових розмірів. Це було зроблено для кращого порівняння розмірів кожного кластера, враховуючи великі відмінності. У порівнянні з результатами сильно зв’язаного компонента і зв’язаного компонента, кластерів набагато більше. Виявлено 129 кластерів. Більшість фільмів об’єднуються в три великі кластери. У ці три кластери було включено 4605 фільмів. 116 кластери містили лише один фільм. Сім кластерів містили два фільми, а три — три фільми. Нарешті було три кластери з 4, 8 і 56 фільмів. Хоча було виявлено більше кластерів, 89% з них містили лише один вузол. Це означає, що 89% кластерів не можна використовувати для рекомендації фільмів. Хоча було більше кластерів із більшою кількістю фільмів, цього все одно недостатньо, щоб бути розглянутим для системи рекомендацій.

Причина цього, швидше за все, полягає в кількості зв'язків, що створюють занадто щільний граф. Інший аспект полягає в тому, що алгоритм Лувена в основному базується на вагових зв'язках. Усі ваги тут були встановлені за замовчуванням 1.0, що зменшує ефект алгоритму. Нарешті, був запущений алгоритм поширення мітки. Щоб не зробити процес занадто

складним до обчислень, ітерації були встановлені на 10. Була зроблена ще одна стовпчаста діаграма, яка показувала розміри кластерів, як можна побачити на рисунку 3.10.

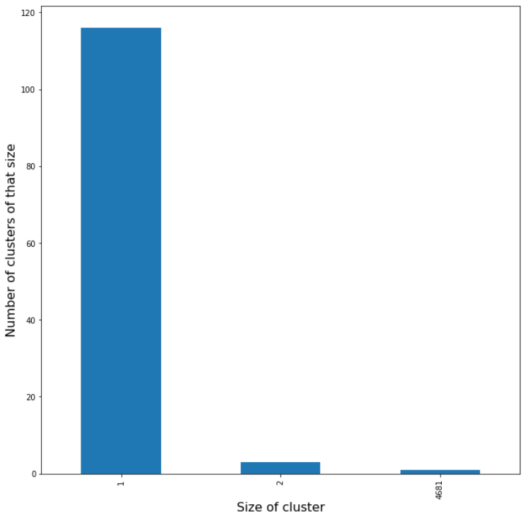

Рис. 3.10. Гістограма, що показує розмір кластерів, знайдених за допомогою алгоритму поширення мітки, і кількість кластерів такого розміру

Цікаво, що функція поширення мітки знайшла кластери такого ж розміру, як зв'язний компонент, і алгоритми сильно зв'язаних компонентів. Ймовірно, це пов'язано з щільністю графу, а також через відсутність ваг. Це, ймовірно, призвело до того, що алгоритм поширення мітки розповсюдив мітки таким концентрованим способом. Від основного кластера могли бути відокремлені лише фільми, які містили дуже різні властивості.

3.1.5 Додані ваги

Алгоритми кластеризації були запущені на зваженій версії графу з однією міткою. При дослідженні зваженої версії цього графу не було потреби розглядати алгоритм сильно зв'язного компонента. Це тому, що алгоритм Neo4j не враховує ваги. Таким чином, результат буде таким же. Результати підрахунку трикутників і коефіцієнт кореляції також будуть такими ж, оскільки структура графу не змінюється. Для кожного з запущених алгоритмів було сформовано 4803 кластери. Тобто кожен фільм формується у власний кластер. Додавання ваг в цю структуру графу значно змінило те, як

працює кожен з алгоритмів. Коли вага не застосовувався, щільність графіка змушувала алгоритми групувати фільми в кілька дуже великих кластерів.

Введення ваг забезпечило необхідну відмінність. Однак щільність графів все ще перешкоджала показу справжніх шаблонів. Кількість ваг між графом також може мати проблеми під час виконання алгоритму. Якщо дивитися на рисунок 3.2 раніше, діапазон зважень між зв'язками був дуже великим. Це, швидше за все, призвело до більшого поділу. Що стосується модульності Лувена, алгоритм максимізує свою передбачувану точність, порівнюючи щільність з ваговими зв'язками. Щільність зв'язків була дуже високою, а найпоширеніші зв'язки, спільні ключові слова, мали дуже низькі коефіцієнти. Алгоритм не міг побудувати кластери при порівнянні цих двох зв'язків.

Агрегований граф

Завдяки реструктуризації графіка кількість зв'язків було зменшено. Отже, це зменшило кількість утворених трикутників, а отже, вплинуло на коефіцієнт кластеризації. Графіки нового трикутника та коефіцієнти кластеризації можна побачити на рисунку 3.11.

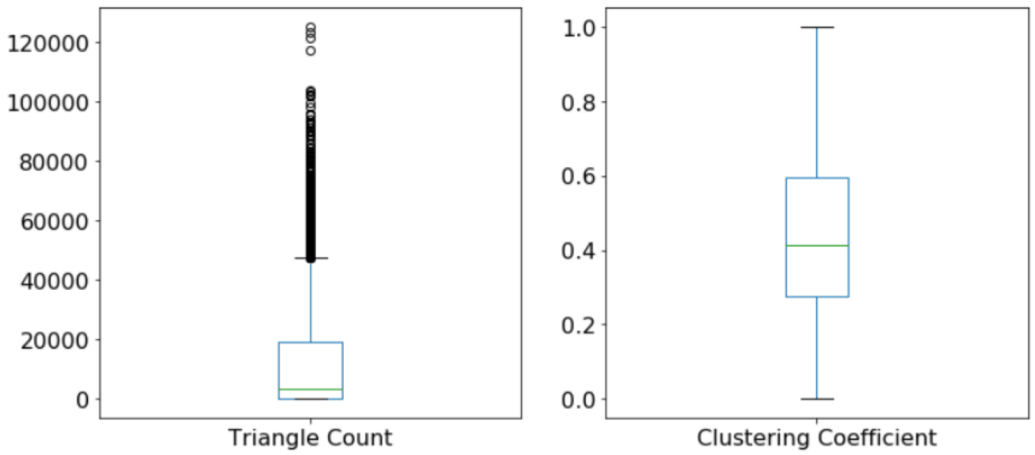

Рис. 3.11. Діаграми кількості трикутників і коефіцієнтів кластеризації.

Як і очікувалося, кількість трикутників було значно зменшено, середнє число трикутників тепер становить 13472,84. В результаті цього середнє значення коефіцієнта кластеризації було в 0,51 рази більше, ніж попередньо

агрегований графік. Це помітне зниження здатності багатьох фільмів до групування. Першим алгоритмом, запущеним тут, був граф зв'язних компонентів. Різна структура зменшує кількість з'єднань і має призвести до більшої кількості поділів. Нові ваги графіка також були враховані.

За допомогою цього алгоритму було виявлено 678 кластерів, проте 4126 з них були в одному кластері, решта фільмів містилися окремо в одному вузлі кластерів. Це далеко не позитивний результат.

3.2 Граф з кількома вузлами

Аналіз розроблених графів із кількома мітками був розділений на дві частини, щоб відповідати першим двом основним цілям. Першою частиною було дослідження побудови графів, а потім застосування алгоритмів на основі спільноти.

3.2.1 Побудова графу

Запити Cypher успішно побудували граф, що містить вузли ACTOR, вузли DIRECTOR, вузли MOVIE і GENRE, а також відповідні зв’язки. Зразок графіка можна побачити на рисунку 3.12.

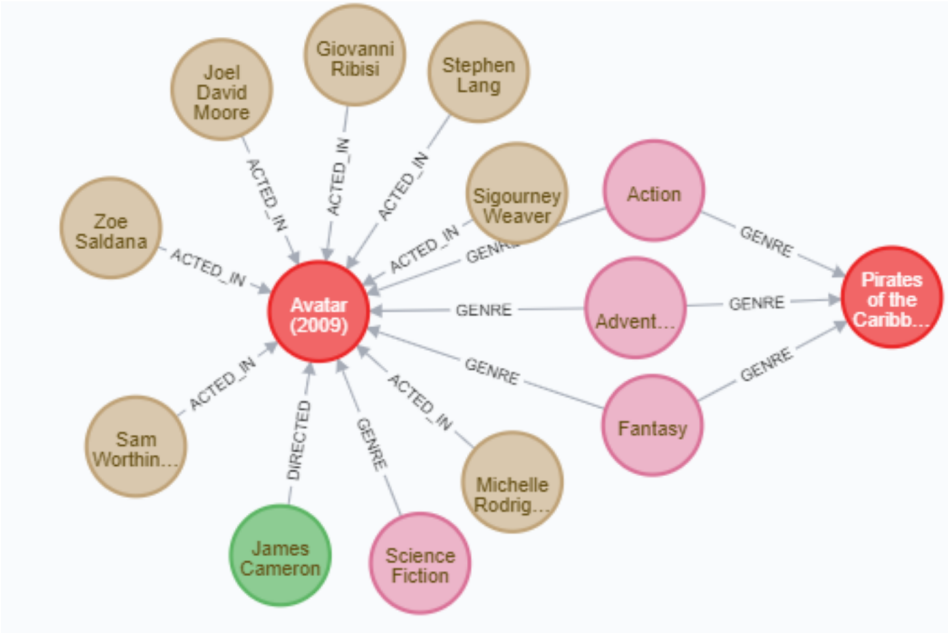

Рис. 3.12. Приклад бази даних із кількома вузлами

Дивлячись на цей рисунок, можна побачити, що потрібні вузли були створені успішно. Також можна побачити, що правильні відносини з'єднують вузол.

Кількість кожного типу вузлів можна побачити нижче в таблиці 3.3

Тип вузла	**Кількість**
ACTOR	13,025
DIRECTOR	2417
MOVIE	4803
GENRE	20

Табл. 3.3. Таблиця з кількістю кожного типу вузла.

Тут видно, що всі 4803 фільми в базі даних були додані до графіка, а також усі 20 жанрів. Було додано 13 025 акторів. Це 3,20 акторів на вузол. Це розумне значення, оскільки зібрано сім акторів з кожного фільму, але вузли для кожного актора різні. Отже, значення акторів має бути менше 7, щоб врахувати перехід у акторському складі. Результат показує, що в середньому фільми поділяють 45,7% найкращих акторів. Кількість кожного типу зв'язків можна побачити нижче в таблиці 3.4

Тип зв’язку	**Кількість**
ACTED_IN	32791
DIRECTED	4775
HAS_GENRE	12160

Табл. 3.4. Кількість зв’язків кожного типу

Існує 4803 вузли MOVIE, і відносини ACTED_IN були засновані на семи найкращих акторах у цьому фільмі. Це означало б, що має бути 33621

спільних акторських відносин. Проте існує лише 32791 стосунків ACTED_IN на MOVIE. Це в середньому 6,827 акторів на фільм. Хоча це дуже близько до прогнозованого, невелика розбіжність свідчить про те, що сталося щось несподіване. Було 650 випадків, коли у фільм було призначено менше семи акторів. Ймовірно, це пов'язано не з процесом імпорту, а з тим, що деякі фільми в базі даних фільмів TMdB5000 містять менше семи акторів.

Аналогічно, було 4775 зв'язків DIRECTED_BY, тобто 0,994 режисера на фільм. Очікується, що на один фільм буде трохи більше 1 режисера, оскільки всі, крім кількох фільмів, мають лише одного режисера. Різниця у відносинах знову дуже незначна і, швидше за все, через брак даних про режисера деяких фільмів у базі даних фільмів TMDB 5000. У зв'язках HAS_GENRE було 2,532 зв'язку на вузол фільму. Таку цінність очікується, оскільки тенденція до фільмів передбачає від 2 до 3 режисерів на фільм. Якщо порівняти кількість зв'язків із кількістю вузлів, то на вузол ACTOR припадає 2,48 відносин ACTED_IN. Це означає, що середній актор займає перше місце в 2,48 фільмах. Це половина від кількості фільмів, у яких бере участь середній актор, яка була знайдена в результаті досліджень. Це означає, що половина ролей акторів із баз даних фільмів TMDb 5000 увійшли до семи найкращих. Це також свідчить про те, що кожен режисер знімає в середньому 2.00 фільмів. Це узгоджується з дослідженням. Результати аналізу цього графу свідчать про те, що база даних фільмів TMDb 5000 була успішно адаптована у формат бази даних графів, що підтверджує першу мету цього дослідження.

3.2.2 Дослідження кластеризації

Оскільки структура цього графіка не дозволяла трикутникам формувати алгоритми на основі трикутників, кількість трикутників, коефіцієнт зв'язності, компонент зв'язку та компонент сильної зв'язку були непридатними. Таким чином, кластеризаційний аналіз проводився з використанням поширення мітки та модульності Lovain. Цей графік також не

мав ваг у своїх відносинах, тому аспект алгоритмів кластеризації також не міг бути досліджений.

Розмноження Лувена

Кількість фільмів у кожному кластері після зйомки режиму можна побачити на рисунку 3.13.

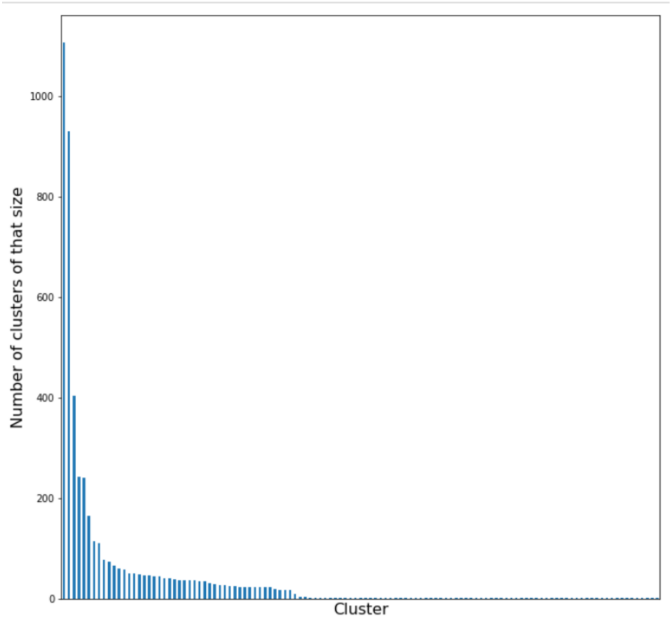

Рис. 3.13. Гістограма, що показує кількість фільмів на кластер за допомогою модульності Лувена.

Цей алгоритм створив 678 кластерів із середнім показником 40,36. Як видно на цьому зображенні, у вузлах домінують в основному кілька стовпців, як і з результатами кластеризації однієї мітки. Як видно на цьому зображенні, у вузлах домінують в основному кілька стовпців, як і з результатами кластеризації однієї мітки. Найбільший кластер містив 1106 фільмів. Це означає, що в одному кластері існує 0,23 фільмів. Було 58 кластерів, які містили лише один фільм. При порівнянні цих результатів із викладеними показниками кластеризації. Метод Лувена погано працює з цією структурою графіка як рекомендаційною системою. Кластери не утворюють рівномірного розподілу кластерів, що містять усі фільми.

Є занадто багато фільмів, які не можна рекомендувати через те, що вони існують у власному кластері. Також важко дати значущі рекомендації, коли чверть усіх фільмів вписується в один кластер.

Поширення міток

Той самий метод був застосований до графу з поширенням мітки. Результати можна побачити нижче на рисунку 3.14

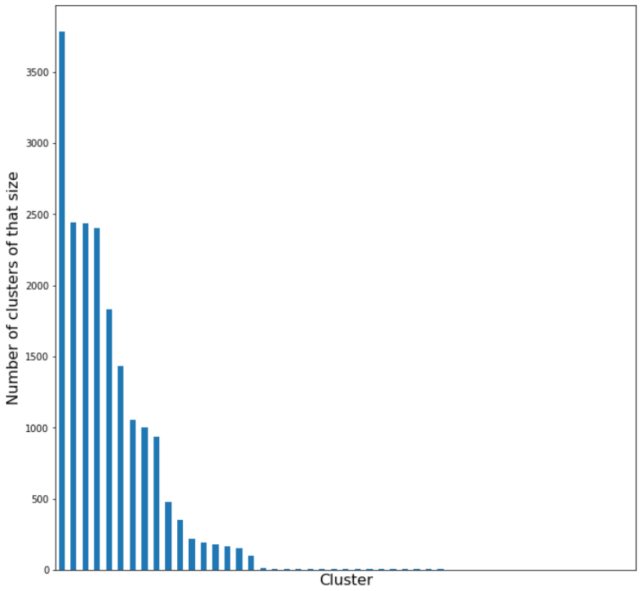

Рис. 3.14. Гістограма, що показує кількість фільмів на кластер за допомогою поширення міток.

Найдивовижніше, якщо дивитися на цю стовпчасту діаграму, це те, що розмір кластерів в сумі набагато більше, ніж кількість фільмів. Це очікувано, оскільки кластерів було набагато більше, ніж у фільмі. Варто провести подальше дослідження, щоб визначити, чи розподіляються фільми рівномірно. Метою цього експерименту було дослідження окремо згрупованих фільмів. Хоча, можливо, варто додатково дослідити цю форму кластеризації як метод рекомендації фільму. Навіть враховуючи це, розподіл розміру кластерів занадто великий, щоб побудувати успішну систему рекомендацій, як було зазначено раніше.

3.3 Порівняння графів

Якщо порівняти кластеризацію двох графів, стає ясно, що жоден із використаних підходів не був успішним у створенні окремих кластерів. Однак найбільш близькою до ефективної виявилася модуляція Ловейна на графіку Multi-Label. Хоча неможливо було повністю дослідити, як зважена

частота документа порівнялася б із показниками кластеризації, оскільки не було достатньо обчислювальної потужності для повного дослідження.

ВИСНОВКИ

Двома основними цілями цього дослідження були розробка бази даних графів для представлення даних фільму та застосування до неї алгоритмів кластеризації для пошуку кластерів, які потім можна було б використовувати для системи рекомендацій. Перша мета була успішною, оскільки було побудовано багато різновидів графіків. Друге завдання виявилося не таким успішним. Хоча було багато різних підходів, був знайдений метод кластеризації бази даних графів, щоб її можна було використовувати для системи рекомендацій. Після застосування кількох алгоритмів кластеризації жодного результату не вдалося задовольнити метрикам, які були встановлені, щоб вважати метод придатним для рекомендаційної системи.

Neo4j виявився ефективним інструментом для побудови графіків і застосування алгоритмів кластеризації. Був використаний для успішної розробки графічної бази даних з описом фільмів. Більшість інформації з бази даних фільмів TMDb 5000 успішно перенесено в базу даних графів. Проте велика частина описаних даних не підходила для графічної бази даних. Не знайдено жодного методу, який міг би включати бюджет фільму, будь-яку інформацію про слоган, огляд, бюджет, дохід чи час показу. Будь-яка з цієї інформації може виявитися корисною при використанні її для розробки рекомендаційної системи. Однією з проблем, яка виникла, була відсутність обчислювальної потужності, яка перешкоджала розробці більш складних ваг. Подальше дослідження різних методів зважування термінів частоти було б корисним, якщо є доступ до більш продуктивних комп’ютерів.

Розглядаючи побудову графіків, як базовий, невагомий одновузловий граф, так і графи з кількома мітками були успішно побудовані за допомогою Neo4j. Обидва ці графіки можна було запитувати за допомогою мови запитів Neo4j Cypher. Хоча графік з одним вузлом відображав більше інформації про кожен фільм у його вузлах і зв’язках.

Граф з одним вузлом можна було адаптувати для додавання ваги зв’язкам. Це неможливо було зробити за допомогою графу з кількома

мітками. Проте були запропоновані потенційні розробки для подальшого вивчення цього. Через це немає жодних подальших розробок багатомітного графа. Однак для графа окремих вузлів три графи, які були успішно побудовані в Neo4j, які містили всю інформацію про базу даних TMBd 5000 Movie Database, і один, який успішно створив вибірку. Виходячи з цих результатів, для розробки баз даних графів краще використовувати граф з однією міткою.

На відміну від графа з кількома мітками, на графі з однією міткою можна було виконати всі обговорювані алгоритми для всіх, окрім терміну, через його розмір. Хоча запропоновані потенційні зміни дозволять це зробити. Однак це був результат як поширення мітки, так і модульності Лувена, який працював краще, ніж будь-який результат у графі єдиної мітки при використанні рівного розподілу як метрики. Хоча це ще не може бути по-справжньому підтверджено, оскільки кластери містять не лише вузли, які представляють фільми. Лише групування цих вузлів можна використовувати для дослідження рекомендацій фільму.

У цій роботі досліджується кластеризація грфів фільмів з кінцевою метою розробки системи, яку можна використовувати як систему рекомендацій. Однак через те, що застосовувані методи кластеризації не дають результатів, які задовольняють встановленим показникам, не було можливості експериментально визначити, чи будуть ці рекомендаційні фільми в одному кластері працювати як функціональна рекомендаційна система. Це можна зробити шляхом аналізу другого набору даних, який містить оцінки користувачів фільмів. Це було зроблено за допомогою даних, зібраних з рецензій на фільми людей, використання фільмів, оцінених людиною позитивно, і визначення їх ідеального кластера, ідеальним набором даних є дані MovieLens []. Дані містять ідентифікатор користувача, назву фільму, оцінку 1–5 із кроком 0,5, яка описує задоволення від фільму. Оцінка 5 означає максимальне задоволення, а оцінка 1 означає мінімальне задоволення. Позитивні фільми, визначені як оцінка 3+, можуть бути

показником фільму, який сподобався. Кластер з найбільшою кількістю позитивних оцінок користувачів буде використовуватися як кластер рекомендацій. Для експериментального дослідження цього слід використовувати розділення тестового поїзда.

Хоча результати дослідження кластеризації для рекомендацій фільмів були непереконливими, у цій області ще багато можливої роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Al Hasan, M., Chaoji, V., Salem, S., and Zaki, M. (2006). Link prediction using supervised learning. In SDM06: workshop on link analysis, counter-terrorism and security.
2. Angles, R. and Gutierrez, C. (2008). Survey of graph database models. ACM Computing Surveys, 40
3. Assefi, M., Liu, G., Wittie, M. P., and Izurieta, C. (2015). An experimental evaluation of apple siri and google speech recognition. Proccedings of the 2015 ISCA SEDE
4. Bar-Ilan, J. and Peleg, D. (1991). Approximation algorithms for select- ing network centers.
5. Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment, 2008
6. Bobadilla, J., Ortega, F., Hernando, A., and Bernal, J. (2012). A collaborative filtering approach to mitigate the new user cold start problem.
7. Brusilovsky, P. and Millán, E. (2007). User models for adaptive hypermedia and adaptive educational systems.
8. Buerli, M. and Obispo, C. (2012). The current state of graph databases.
9. Bunch, C., Chohan, N., Krintz, C., and Shams, K. (2011). Neptune: a domain specific language for deploying hpc software on cloud platforms.
10. Businessweek, B. (2013).
11. Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.-Y., and Moon, S. (2009). Analyzing the video popularity characteristics of large-scale user generated content systems.
12. Chung, H., Park, J., and Lee, S. (2017). Digital forensic approaches for amazon alexa ecosystem. Digital Investigation
13. Clauset, A., Newman, M. E., and Moore, C. (2004). Finding community structure in very large networks.
14. Devine, P. G. (1989). Stereotypes and prejudice: Their automatic and controlled components.
15. Dial, R. B. (1969). Algorithm 360: Shortest-path forest with topological ordering [h]
16. Duda, R. O. and Hart, P. E. (2001). Dg stork pattern classification. John Wiely and Sons.
17. Fatemi, M. and Tokarchuk, L. (2013). A community based social recommender system for individuals & groups.

18. Fleischer, L. K., Hendrickson, B., and Pınar, A. (2000). On identifying strongly connected components in parallel
19. Follows, S. (2013). How many films in an average film career.
20. Fouss, F., Yen, L., Pirotte, A., and Saerens, M. (2006). An experimental investigation of graph kernels on a collaborative recommendation task.
21. Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms.
22. GroupLens (2018). Movielens 20m dataset.
23. Gubichev, A., Bedathur, S., Seufert, S., and Weikum, G. (2010). Fast and accurate estimation of shortest paths in large graphs. In Proceedings of the 19th ACM international conference on Information and knowledge management
24. Heckemann, R. A., Hajnal, J. V., Aljabar, P., Rueckert, D., and Ham- mers, A. (2006). Automatic anatomical brain mri segmentation combining label propagation and decision fusion. NeuroImage
25. Holzschuher, F. and Peinl, R. (2013). Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In Proceedings of the Joint EDBT/ICDT 2013 Workshops
26. Huang, Z., Chung, W., Ong, T.-H., and Chen, H. (2002). A graph-based recommender system for digital library
27. Kannan, R., Vempala, S., and Vetta, A. (2004). On clusterings: Good, bad and spectral.
28. Karlgren, J. (1994). Newsgroup clustering based on user behavior-a recommen- dation algebra.
29. Kemper, C. (2015). Beginning Neo4j. Springer.
30. Kleinberg, J. M. (2002). Small-world phenomena and the dynamics of information.
31. Lakiotaki, K., Matsatsinis, N. F., and Tsoukias, A. (2011). Multicriteria user modeling in recommender systems.
32. Lalwani, D., Somayajulu, D. V., and Krishna, P. R. (2015). A community driven social recommendation system.
33. Lam, X. N., Vu, T., Le, T. D., and Duong, A. D. (2008). Addressing cold-start problem in recommendation systems.
34. Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks.
35. Schaeffer, S. E. (2007). Graph clustering. Computer science review
36. Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-commerce recommendation applications. Data mining and knowledge discover

37. Smith, B. and Linden, G. (2017). Two decades of recommender systems at amazon. com.
38. Goldberg, D., Nichols, D., Oki, B., and Terry, D. 1992. Using collaborative filtering to weave an information tapestry.
39. Goldberg, D., Nichols, D., Oki, B., and Terry, D. 1992. Using collaborative filtering to weave an information tapestry. Communications of the ACM 35, 12, 61–70.
40. Bonhard, P. 2004. Improving recommender systems with social networking. In Pro- ceedings Addendum of the 2004 ACM Conference on Computer-Supported Cooperative Work. Chicago, IL, USA.
41. Shardanand, U. and Maes, P. 1995. Social information filtering: Algorithms for au- tomating “word of mouth”. In Proceedings of the ACM CHI Conference on Human Factors in Computing Systems. ACM Press, Denver, CO, USA, 210–217.
42. Herlocker, J., Konstan, J., and Riedl, J. 2002. An empirical analysis of de- sign choices in neighborhood-based collaborative filtering algorithms. Information Re- trieval 5, 4, 287–310.
43. Sowa, J. F. (1976). Conceptual graphs for a data base interface.
44. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. (2010). A comparison of a graph database and a relational database: a data provenance per- spective.
45. Mark Needham, A. E. H. (2019). Graph Algorithms: Practical Examples in Apache Spark Neo4, volume 1 of 1. O’Reilly, 1 edition.
46. Mojo, B. O. (2019). 2019 studio market share.