НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

УДК: 004.021

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___” _____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему: «*Комплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі*»

Виконала: студентка *4* курсу, групи *ІС-51*

Борисенко Анастасія Димитріївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Попенко В.Д.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Тєлишева Т.О.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц. каф. АУТС, к.т.н., доц. Репнікова Н.Б.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка Борисенко А.Д. _____
(підпис)

Київ – 2019 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 12 рисунків, 8 таблиць, 1 додаток, 23 джерело.

Дипломний проект присвячений розробці комплексу задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальних мережах.

Метою роботи є сприяння глибшому аналізу соціальних мереж шляхом розробки застосування для пошуку спільнот користувачів у соціальних мережах.

У розділі з інформаційного забезпечення були визначені вхідні та вихідні дані до застосування, була розроблена структура бази даних.

Розділ математичного забезпечення присвячений опису існуючих методів. Обрані два з них, як методи вирішення рекомендаційної системи.

У розділі з програмного забезпечення описані основні засоби розробки застосування, висунуті вимоги до технічного забезпечення, обрано та обґрунтовано архітектуру програмного забезпечення.

У технологічному розділі описані інструкція користувача та методи випробування програмного продукту.

РЕКОМЕНДАЦІЙНА СИСТЕМА, АНАЛІЗ СОЦІАЛЬНИХ МЕРЕЖ, СОЦІАЛЬНИЙ ГРАФ.

ABSTRACT

Structure and scope of work. Explanatory note of the diploma project consists of five sections, containing 12 figures, 8 tables, 1 supplement, 23 sources.

The diploma project is devoted to the development of a set of tasks for comparative analysis of algorithms for the detection of hidden subgroups in social networks.

The goal of the work is to promote a deeper analysis of social networks by developing applications for searching community of users in social networks.

In the section on information provision input and output data were identified prior to application, a database structure was developed.

The section of mathematical support is devoted to the description of existing methods. Two of them are selected as methods of solving the advisory system.

The software section describes the main tools for developing the application, the requirements for technical support, the architecture of the software is selected and grounded.

The technology section describes the user manual and test methods for the software product.

RECOMMENDATION SYSTEM, ANALYSIS OF SOCIAL NETWORKS, SOCIAL GRAPH.

ЗМІСТ

ВСТУП

Графи вкрай зручні для представлення широкого спектра систем в різних областях. Біологічні, соціальні, технологічні та інформаційні мережі можу бути представлені в якості графів, і тому аналіз графів став ключовим механізмом для вивчення властивостей цих систем. Наприклад, аналіз соціальних мереж бере початок в 30-х роках минулого століття, і на цей момент став одним з найважливіших напрямків в соціології [1].

Графові уявлення реальних систем відрізняються своєю неоднорідністю. В особливих групах вершин концентрація дуг висока, в той час як між цим групами вона невелика. Це властивість реальних мереж називається структурою спільнот [2] або кластеризаціею. Спільнота, або кластер, - це набір вершин, відносно сильно пов'язаних один з одним, і, можливо, що вони володіють загальними властивостями і / або грають схожі ролі в мережі. На рисунке 0 представлений приклад графа з виділеними спільнотами.

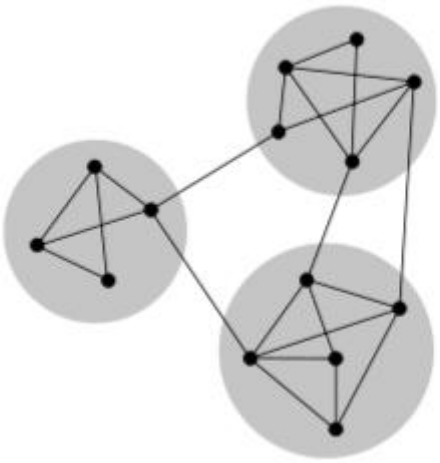

Рисунок 0 – Граф з трьома спільнотами

Суспільство пропонує велику кількість можливих спільнот: сім'я, колеги, друзі, жителі одного міста, громадяни одного державиі та ін. Широке поширення Інтернету призвело до появи великої кількості віртуальних груп, онлайн спільнот. У мережах взаємодій білок-білок спільнотами можна

назвати групи протеїнів, що мають однакові функції в клітині [3, 4]. У графі Всесвітньої Павутини спільноти можуть відповідати наборам сторінок, присвячених загальній або пов'язаним темам.

Наведу кілька прикладів конкретних застосувань спільнот. Кластеризація веб-клієнтів по їх інтересам і географічному положенню дозволяє поліпшити продуктивність сервісів в Інтернеті, кожен отриманий кластер може бути оброблений окремим допоміжним сервером [5]. Кластери великих графів можуть бути використані при створенні зручних структур даних для ефективного зберігання таких графів і обробки навігаційних запитів, наприклад, пошуку шляху з одного вузла в інший [6]. Мережі, які самі конфігуруються, структура яких часто змінюється (наприклад, мережі мобільного зв'язку), як правило, не мають централізованої таблиці маршрутизації. Угруповання вузлів в кластери дозволяє швидко генерувати компактні локальні таблиці, що забезпечують ефективну комунікацію [7].

Визначення спільнот і їх кордонів дозволяє класифікувати вузли графа по роду їх діяльності в кластері. Вузли в центральних позиціях, які мають велику кількість зв'язків з іншими членами групи, несуть регулюючі і стабілізуючі функції. А ті вузли що знаходяться на кордонах спільнот є сполучними, і відповідальні за взаємодію з іншими кластерами. Наведена класифікація притаманна різного роду соціальним мережам [8, 9].

Ще один важливий аспект, пов'язаний з кластерної структурою, – ієрархічна організація більшості реальних мереж. Такі мережі зазвичай складаються з спільнот, які в свою чергу містять більш дрібні спільноти. Як приклад можна привести мережу, яка відображатиме будову людського організму: людина складається з органів, органи з тканин, тканини з клітин і ін. Іншим прикладом можуть служити фірми, організовані в піраміду із співробітників, робочих груп, відділів і ін.

Метою пошуку спільнот в графах є визначення кластерів і, по можливості, їх ієрархії, використовуючи лише інформацію про структуру цього графа.

Дипломний проект присвячений розробці комплексу задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальних мережах.

Практичне значення одержаних результатів. Розроблено алгоритми виявлення прихованих підгруп в соціальних мережах.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

На даний момент існує понад 200 широковідомих, і величезна кількість менш популярних сайтів соціальних мереж. У цих мережах зареєстровані мільйони людей, кожен з яких пов'язаний з десятками або навіть сотнями інших користувачів. Всі ці складні взаємозв'язки можна зручно представити у вигляді соціального графа користувачів.

Однією з проблем, що виникають при аналізі соціальних мереж (як, втім, і багатьох інших реальних мереж), є спрямованість зв'язків між вузлами. Наприклад, в графі Всесвітньої Павутини вершинами є сторінки, а зв'язками - посилання з одних сторінок на інші. Ці посилання спрямовані - якщо ми може перейти по посиланню зі сторінки А на сторінку В, то далеко не завжди ми можемо виконати зворотну дію. Пошук спільнот в веб-графі може допомогти визначити штучні кластери, створені фермами посилань з метою поліпшити значення PageRank для певних сайтів і підвищити їхню позицію в пошуковій видачі Google. Таким чином, облік напрямки зв'язків може істотно поліпшити якість поділу на кластери і надати додаткову цінну інформацію про систему. Однак, з метою спрощення алгоритму пошуку спільнот, інформацію про направлення зв'язків часто опускають.

На жаль, спрямованість зв'язків в графі не єдина проблема при роботі з реальними графами. У багатьох таких мережах вузли можуть ставитися більш ніж до одного кластеру. У цьому випадку говорять про пересічних спільнотах. Класичним прикладом знову ж є соціальні мережі, де людина зазвичай належить до декількох груп одночасно: колеги, сім'я, схожі спортивні уподобання і т.д. Традиційні алгоритми виділення спільнот зіставляють кожну вершину тільки одного кластеру, таким чином, втрачаючи потенційно важливу інформацію. Вершини, що належать більш ніж однієї спільноти, часто грають важливу роль посередника між різними секціями

графа. У той же час варто враховувати, що пошук пересічних співтовариств обчислювально набагато складніше завдання, ніж звичайна кластеризація.

Остання важлива проблема в обробці соціальних графів, про яку хотілося б згадати, - це необхідність ефективної роботи з величезним обсягами даних. Наприклад, в мережі Twitter більше 250 мільйонів користувачів і 15 мільярдів зв'язків між ними, а в Facebook понад 500 мільйонів користувачів і 50 мільярдів зв'язків.

1.1.1 Опис процесу діяльності

Об’єктом автоматизації є процес дослідження ефективності роботи алгоритмів пошуку прихованих підгруп у соціальній мережі. А саме побудова порівняльних графіків двох алгоритмів за різними критеріями якості для визначення оптимального методу.

Перед проектуванням діаграми потрібно виділити акторів застосунку. Актором є дослідник. Під актором дослідник будемо розуміти користувача продукту.

Розглянемо процес діяльності користування застосунком за допомогою діаграми бізнес-процесу, зображеної на рисунку 1.1.

Користувач продукту при запуску застосунку може обрати наступні пункти, за допомогою яких він буде перенаправлен на інші віконні форми:

– «Застосування реальних даних», де він має змогу обрати один з двох алгоритмів і подивитись результат його роботи;
– «Порівняння алгоритмів», де користувач має змогу подивитись порівняльні графіки на різних критеріях якості.

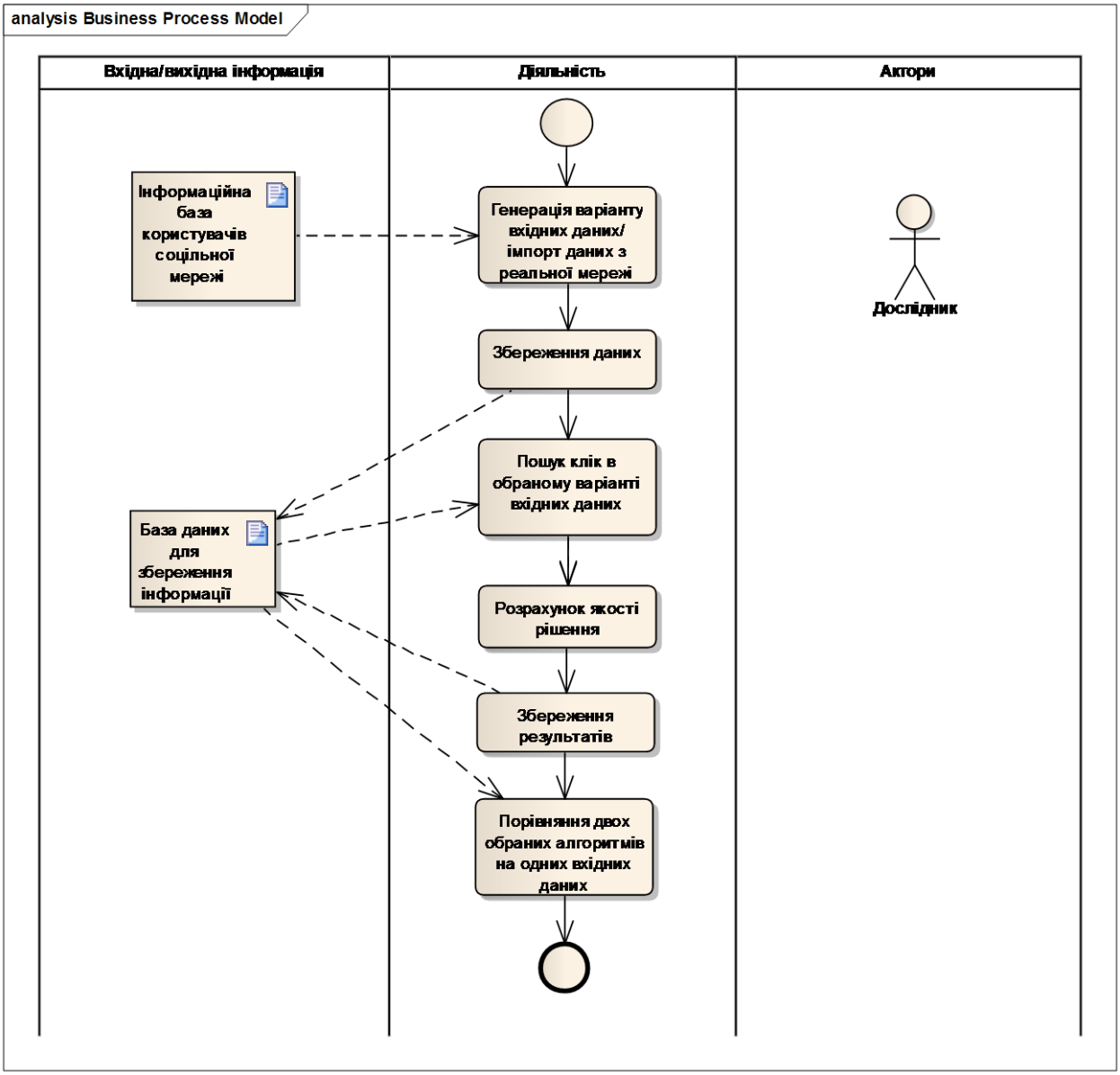

Рисунок 1.1 – Схема структури бізнес-процесу

1.1.2 Опис функціональної моделі

Специфікацію функціональної поведінки системи представимо у вигляді діаграми варіантів використання. Схему структурну варіантів використання наведено у графічному матеріалі.

Актором системи є дослідник (користувач продукту).

Дії або варіанти використання, що виконує в системі актор, наведені в таблиці 1.1, в якій наявний опис актора, варіантів використання та опису дій варіантів використання.

Таблиця 1.1 – Типи залежностей між варіантами використання

Актор	Варіант використання	Опис дії варіанту використання
Користувач	Генерація варіанту вхідних даних/ імпорт даних з реальної мережі	Користувач має змогу згенерувати вхідні дані самостійно або загрузити дані з реальної мережі
	Розрахунок якості рішення	Користувач має змогу переглянути результати розрахунку якості рішення
	Вибір алгоритмів	Користувач має змогу обрати алгоритм пошуку
	Порівняння двох обраних алгоритмів на одних вхідних даних	Користувач має змогу переглянути результати порівняння обраних алгоритмів

1.2 Огляд наявних аналогів

Провівши аналіз ринку, було з’ясовано що прямих конкурентів система не має, так як особливістю системи є те, що в увагу береться переважно порівняння алгоритмів пошуку прихованих підгруп у соціальній мережі на різній кількості даних та за різними критеріями. Проте існують системи, які використовують дані алгоритми саме для знаходження підгруп, але не аналізують їх ефективність.

«Cfinder» програма реалізує всього один метод для виявлення пересічних кластерів в великих мережах, який може бути застосований як до спільнот, так і до білкових з'єднань. Clique Percolation Method (CPM), який

використовується в «CFinder», призначений для виявлення спільнот в невиважених, неорієнтованих мережах. Розширена версія алгоритму, яка включена в «CFinder», може також обробляти зважені і спрямовані мережі.

Фірма «Cambridge Analytica», використовує технології глибинного аналізу даних (зокрема, даних соціальних мереж) для розробки стратегічної комунікації в ході виборчих кампаній в Інтернеті. «Cambridge Analytica» займається збиранням даних про користувачів Інтернету і соцмереж, складанням їх психологічних портретів і розробці персоналізованої реклами. Саме за допомогою цього аналізу «Cambridge Analytica» у 2016 році допомогла Дональду Трампу виграти президентські вибори.

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням розробки даного функціоналу програмного забезпечення є розробка застосування для порівняння двох алгоритмів пошуку спільнот в соціальних мережах за різними критеріями якості.

1.3.2 Цілі та задачі розробки

Метою роботи є сприяння глибшому аналізу соцмереж шляхом розробки застосування для пошуку спільнот користувачів у соціальних мережах.

Для досягнення поставленої мети мають бути вирішені такі задачі:

- Провести аналіз і порівняння існуючих алгоритмів пошуку спільнот на графах;
- Обрати найбільш ефективні і багатообіцяючі з них;
- Керуючись ідеями, реалізованими в обраних алгоритмах, створити програмний засіб, що здійснює пошук спільнот користувачів в соціальних графах.

– Провести тестування і оцінку продуктивності розробленого рішення.

Висновок до розділу

У розділі «Загальні положення» було розлянуто предметне середовище, наведено структуру програми лояльності та її основні принципи.

При описі предметного середовища було визначено та описано процеси діяльності, що представлені у вигляді трьох незалежних процесів, для кожного з яких наведено структурні схеми. Також було створено функціональну модель системи, а саме: побудовано USE-case діаграму, описано акторів системи та визначено функціональні вимоги виконання відповідно до варіантів використання.

Наступним етапом роботи над дипломним проектом став пошук аналогів запропонованого застосунку з подібним функціоналом. Було порівняно функції, які виконують знайдені програмні продукти. Було виявлено декілька систем зі схожим функціоналом, але вони не охоплюють увесь спектр функцій, які запропоновані в дипломному проекті. У дослідженні було сформовано постановку задачі, визначено призначення, мету та задачі розробки. Дипломний проект присвячений роботі розробці інформаційної системи підвищення лояльності споживачів.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Всі вхідні дані інформаційної системи можна розділити на дві категорії:

– вхідні дані для роботи алгоритмів на реальній мережі;
– вхідні дані для роботи алгоритмів на згенерованій мережі для порівняльного аналізу.

Розглянемо першу категорію. Вхідними даними для роботи алгоритмів на реальній мережі є особові дані користувачів соціальною:

– ПІБ;
– дата народження;
– місто проживання;
– місце роботи;
– інтереси користувача.

Вхідними даними другої категорії набір згенерованих LFR – графів, які відрізняються кількістю вершин – від 100 до 1000000. Ці дані будуть записані у базу даних у вигляді пар ID користувачів.

2.2 Вихідні дані

Після роботи алгоритмів на реальній мережі створюється таблиця з особовими даними користувачів, які входять в одну кліку. Назви стовпчиків таблиці представлені у таблиці 2.1.

Таблиця 2.1 – Вихідні дані

ID	Ідентифікатор користувача
Name	ПІБ користувача
Date	Дата народження користувача
City	Місто народження користувача

Продовження таблиці 2.1

Job	Місце роботи користувача
Interests	Інтереси користувача

Після роботи алгоритмів на згенерованій мережі будуються порівняльні графіки двох алгоритмів за такими критеріями:

– кількість користувачів мережі;

– час роботи алгоритму;

– критерій якості алгоритму.

2.3 Опис структури бази даних

Для дипломного проекту було використано СУБД SQLite. Схему бази даних наведено в графічному матеріалі. База даних проекту складається з 7 таблиць, але не всі таблиці пов’язані між собою. Наприклад, для зберігання вхідних даних реальної мережі створено 2 таблиці, які пов’язані між собою:

– «InputData»;

– «Connection».

Для зберігання вхідних даних згенерованої мережі створено 2 таблиці, які пов’язані між собою:

– «Input_Generate»;

– «Connection_Generate».

А для збереження даних, які необхідні для побудови порівняльних графіків роботи алгоритмів створено 3 таблиці, які пов’язані між собою:

– «Data_Result»;

– «Algorithm»;

– «Network».

Опис існуючих таблиць у базі даних наведено у таблиці 2.2.

Таблиця 2.2– Опис таблиць БД

Назва таблиці	Назва стовпця	Тип даних	Призначення
InputData	ID	int	Унікальний ідентифікатор користувача
	Name	char	ПІБ користувача
	Date	date	Дата народження користувача
	City	char	Місто народження користувача
	Job	char	Місце роботи користувача
	Interests	char	Інтереси користувача
Connection	ID_1	int	Ідентифікатор першого користувача
	ID_2	int	Ідентифікатор другого користувача
Input_Generate	ID	int	Унікальний ідентифікатор користувача
Connection_Generate	ID_1	int	Ідентифікатор першого користувача
	ID_1	int	Ідентифікатор другого користувача

Продовження таблиці 2.2

Назва таблиці	**Назва стовпця**	**Тип даних**	**Призначення**
Data_Result	ID	int	Унікальний ідентифікатор
	ID_network	int	Ідентифікатор мережі
	ID_algorithm1	int	Ідентифікатор першого алгоритму
	ID_algorithm2		Ідентифікатор другого алгоритму
	Time_1	double	Час роботи першого алгоритму
	Time_2	double	Час роботи другого алгоритму
	Quality_1	double	Якість рішення першого алгоритму
	Quality_2	double	Якість рішення другого алгоритму
Algorithm	ID	int	Унікальний ідентифікатор алгоритму
	Name	char	Назва алгоритму
Network	ID	int	Унікальний ідентифікатор мережі

Продовження таблиці 2.2

	Number of nodes	int	Кількість вузлів мережі
	Density of connections	double	Щільність зв’язків мережі

При виході з інтерфейсу користувача, всі вихідні дані будуть знищені.

Висновок до розділу

У розділі «Інформаційне забезпечення» були описані вхідні дані, які потрібні для коректної роботи програмного продукту, та вихідні дані, які формуються після обробки алгоритмом вхідних даних.

Описано модель бази даних реальних користувачів, для роботи алгоритмів з реальними даними. Описано модель бази даних згенерованих користувачів, яка використовується для порівняльного аналізу двох алгоритмів. А також описано модель бази даних для збереження результатів порівняння. Було наведено детальний опис полів кожної таблиці.

Описано вигляд виведенної інформації після роботи програмного продукту.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Візьмемо для розгляду звичайну школу і її учнів. Для багатьох дітей найближчими друзями є однокласники (хоча, звичайно ж, часто виникає дружба і між учнями різних класів). Таким чином, можна сказати, що всі учні школи розбивалися на групи, в яких майже всі дружили. Це - класичний приклад спільноти в графі. Знову ж, суворого визначення спільноти немає. Спільнотою називають множину вершин, внутрішні зв'язки якої сильніше, ніж зовнішні. Якщо розглядати класи як спільноти, то вони розбивають всіх учнів на непересічні групи (один учень не може одночасно вчитися в двох класах), які покривають всі вершини в графі школи. Нерідко трапляється так, що, будучи випускниками, шкільні друзі надходять в один університет і продовжують спілкуватися. Тоді, від імені одного з них, при розподілі друзів на університетських і шкільних, другий друг буде потрапляти в обидві групи. Таким чином, соціальний граф може покриватися не тільки непересічними спільнотами; вершини можуть належати кільком співтовариствам відразу.

У даній роботі ми порівняємо два алгоритму виділення спільнот в соціальних графах.

Аналіз роботи алгоритмів зводиться до таких аспектів:

- провести аналіз і порівняння існуючих алгоритмів пошуку спільнот на графах;
- обрати найбільш ефективні і багатообіцяючі з них;
- керуючись ідеями, реалізованими в обраних алгоритмах, створити програмний засіб, що здійснює порівняння алгоритмів пошуку спільнот користувачів в соціальних графах;
- провести тестування і оцінку продуктивності розробленого рішення.

3.2 Математична постановка задачі

Задача виявлення спільнот.

Нехай є мережа G, яка має n вершин та m ребер. Та спільнота C, яка має n_C вершин та m_C ребер.

Розрахуємо щільність графу:

$$\rho = \frac{2m}{\left(n(n-1)\right)}. \tag{3.1}$$

Щільність внутрішніх зв'язків спільнот:

$$\delta_{int}(C) > \rho. \tag{3.2}$$

Щільність зовнішніх зв'язків:

$$\delta_{ext}(C) = \frac{m_{ext}}{\left(n_C(n-n_C)\right)}, \tag{3.3}$$

$$\delta_{ext}(C) < \rho, \tag{3.4}$$

де m_{ext} – кількість зовнішніх ребер.

Завдання виявлення спільнот полягає в максимізації суми різниць щільності $(\delta_{int} - \delta_{ext})$ по всіх спільнотах мережі.

3.3 Обґрунтування методу розв'язання

Розглянемо та порівняємо найбільш вдалі та ефективні алгоритми кластеризації графів з кількох сімейств алгоритмів. На підставі результатів досліджень буде обрано два алгоритма, на базі яких буде розроблена програма.

Перше сімейство алгоритмів – це методи ієрархічної кластеризації. Ієрархічні підходи використовують міру подібності вершин та групують вершини, щоб отримати природне розбиття графа.

Методи ієрархічної кластеризації поділяються на дві основні категорії:

– агломеративні – ітеративне злиття груп вершин з високим ступенем подібності;

– дівізімні - ітеративне видалення ребер між вершинами з малим ступенем подібності.

Широко відомий алгоритм, який використовує ієрархічну кластеризацію, був запропонований Гірван і Ньюменом.[11] Це дівізімний алгоритм, який ітеративно видаляє ребра з високим проміжним показником (betweenness). Проміжну міру можна обчислити як, наприклад, кількість найкоротших шляхів, що пролягають через ребро. Чим вище це значення, тим імовірніше ребро знаходиться поза кластера і є зв'язуючим.

На рисунке 3.1 представлений приклад розподілу проміжної міри по ребрах.

Ієрархічна кластеризація дозволяє отримати хороші результати на реальних даних, але є обчислювально складної і не масштабується.

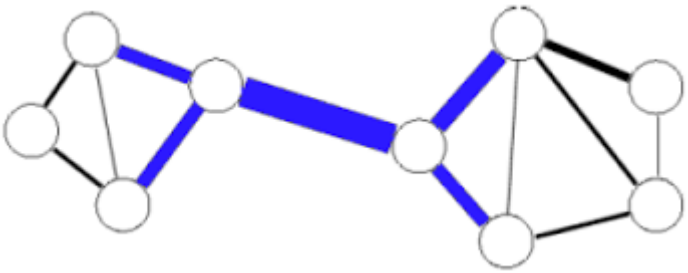

Рисунок 3.1 – Проміжна міра

Друге сімейство алгоритмів – це методи максимізації цільової функції. Алгоритми цієї групи оптимізують значення деякої функції якості розбиття графа на частини. Класичним вибором цільової функції є модулярність (modularity) та її модифікації:

$$Q = \frac{1}{2m}\sum_{c\in P}\sum_{i,j\in c} A[i,j] - \Pr\left(A[i,j] = 1\right), \qquad (3.5)$$

де P – множина спільнот, A – матриця інцидентності.

Ньюмен[11] використовує агломеративний підхід разом з жадібною оптимізацією. Спочатку кожна вершина є кластером, на кожному наступному кроці відбувається злиття кластеров, яке максимально збільшує показник модулярності.

Модулярність запроваджується як характеристика розбиття графа на непересічні компоненти.

Третє сімейство алгоритмів – це методи лінійних графів. Алгоритм лінійних графів[12], по суті, полягає в кластеризації ребер. Можна виділити три основні кроки алгоритму:

1. перетворення початкового графа в лінійний:
   – кожне ребро стає вершиною в лінійному графі;
   – кожна вершина стає клікою в лінійному графі.
2. розбиття лінійного графа на частини алгоритмом для пошуку непересічних спільнот;
3. зворотне перетворення отриманих спільнот лінійного графа в спільноти початкового графа.

Приклад перетворення ділянки графа в лінійний представлений на рисунке 3.2.

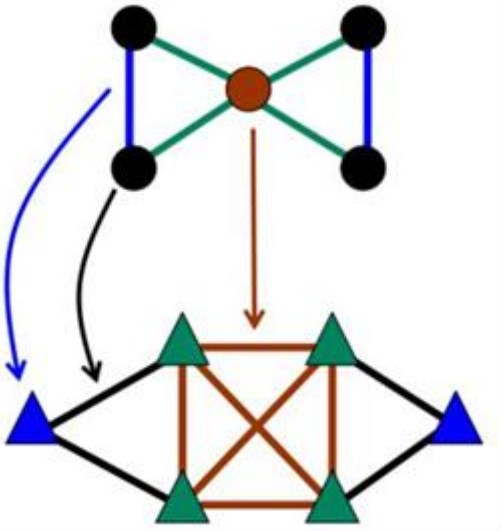

Рисунок 3.2 – Перетворення в лінійний граф

Четверте сімейство алгоритмів – це локальні методи. При локальній роботі з кластерами (не беручи до уваги структуру цілого графа) часто розглядається відношення кількості внутрішніх та зовнішніх ребер або трикутників кластера[13]. Крім того, спільноти можуть складатися з наборів пов'язаних клік (повний підграф графа) різного розміру[14, 15].

Саме з четвертого сімейства було обрано два алгоритми для порівняльного аналізу. А саме: перколяція клік (k-Clique percolation) та жадібний алгоритм розширення клік (Greedy Clique Expansion).

3.4 Опис методів розв’язання

Перколяція клік (Clique percolation).

Clique percolation method [14] будує спільноти з суміжних k-клік (повний підграф графа, який складається з k вершин). Дві k-кліки звуться суміжними, якщо вони містять k-1 спільних вершин. Спільнота визначається, як максимальне об’єднання k-клік, будь-які дві з яких з’єднанні набором суміжних k-клік. Таке визначення спільноти допускає перетин природним чином.

У поліпшеному варіанті класичної перколяції [15] для пошуку спільнот використовується двочастковий граф. Вузлами цього графа можуть бути або k-кліки, або k-1-кліки. K-кліка пов'язана з k-1-клікою, тільки якщо остання є підклікой цієї k-кліки. Таким чином, пов'язані непересічні компоненти цього графа є кінцевими спільнотами.

На рисунке 3.3 наведено приклад знайденого покриття графа при k = 4.

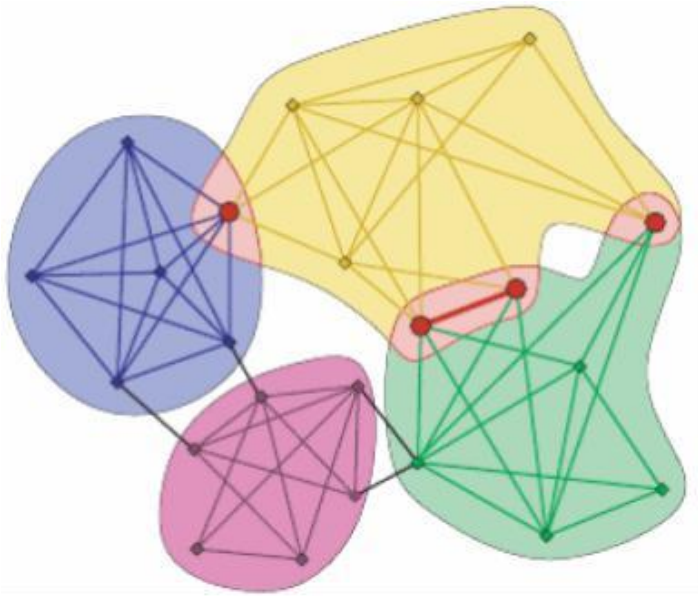

Рисунок 3.3 – Результат роботи методу перколяції

Жадібний алгоритм розширення клік (Greedy Clique Expansion).

Цей локальний алгоритм намагається оптимізувати оціночну функцію для кожної конкретної спільноти. На початку роботи алгоритм обирає початкові «ядра» спільноти, як правило, це вершини або трикутники графа. Потім запускається цикл, який складається з наступних кроків:

– обирається найбільша незакінчена спільнота;
– до цієї спільноти послідовно додаються вершини-сусіди, які максимізують приріст оціночної функції спільноти, до тих пір, поки цей приріст не стане від'ємним;
– якщо оціночна функція досягла максимум, спільнота вважається завершеною, і алгоритм переходить на новий виток.

Одним з головних покращень [13] є вибір максимальних клік в якості «ядер» спільнот. Саме цей вибір є одною з ключових частин алгоритму. Незважаючи на те, що ця задача є обчислювально складною, кліки можуть бути найдені досить швидко в розріджених графах, якими і є соціальні мережі. Автори метода GCE [13] пропонують використати для цієї задачі алгоритм Брона-Кербоша [16], як один з найефективніших. Томита et al. в [17] показали, що в найгіршому випадку складність алгоритму складатиме $O(3^{n/3})$, де n – кількість вершин в графі.

Нижче представлений лістинг модифікованого алгоритму Брона-Кербоша, який буде використаний в подальшому. ***clique*** - множина, що містить на кожному кроці кліку (необов'язково максимальну), що відповідає цьому кроці, ***CAND*** - множина вершин-кандидатів на потрапляння в ***clique***, ***FINI*** - множина вершин, які вже використовувалися для розширення ***clique*** на попередніх ітераціях циклу, ***Γ(v)*** - вершини-сусіди для ***v***.

```
method dfs(CAND, clique)
  FINI = {}
  while CAND \ FINI != ∅ do
    v = argmax(|Γ(v) ∩ CAND \ FINI|)
    clique ∪= {v}
    CAND' = Γ(v) ∩ CAND \ FINI
    if CAND' = ∅ then
      emit clique
    else
      dfs(CAND', clique)

    CAND \= {v}
    FINI ∪= CAND'
    clique \= {v}
```

На рисунках 3.4 та 3.5 представлені приклад графа для пошуку клік та схема роботи алгоритму на цьому графі.

Рисунок 3.4 – Приклад графа для пошуку максимальних клік

Рисунок 3.5 – Схема роботи алгоритму пошуку максимальних клік

Квадратом відзначені вершини, додані в кліку на відповідних кроках. Чорними колами відзначені вершини, складові ***FINI*** на відповідному кроці.

Як оціночної функції спільноти, значення якої оптимізує алгоритм, візьмемо локальну функцію адекватності спільноти S:

$$F_S = \frac{k_{in}^S}{(k_{in}^S + k_{out}^S)^\alpha}, \tag{3.6}$$

де k_{in}^S – внутрішня ступінь спільноти S, k_{out}^S – зовнішня ступінь спільноти S, α – параметр. Інакше кажучи, k_{in}^S – це подвоєне число ребер, обидві вершини яких знаходяться в S (сума ступенів усіх вершин в S), а k_{out}^S – число ребер, тільки одна вершина яких належить S.

Розширення клік вимагає знання сусідів спільноти (назвемо безліч сусідів спільноти S фронтом $f(S)$). На кожному наступному кроці вибирається вершина v_{max} з фронту, яка максимально збільшує значення адекватності для спільноти, і додається до цієї спільноти. Фронт змінюється наступним чином:

$$f(S \cup v_{max}) = (f(S) \cup \Gamma(v_{max}) \backslash S) - \{v_{max}\}, \tag{3.7}$$

де $\Gamma(v_{max})$ – множина сусідів вершини v_{max}. При кожному оновленні фронту значення k_{in}^S і k_{out}^S також оновлюються:

$$k_{in}^{S \cup v_{max}} = k_{in}^S + 2|S \cap \Gamma(v_{max})|, \tag{3.8}$$

$$k_{out}^{S \cup v_{max}} = k_{out}^S + |\Gamma(v_{max})| - 2|S \cap \Gamma(v_{max})|. \tag{3.9}$$

При розширенні клік не виключені випадки, коли спочатку різні кліки розширюються в одну і ту ж спільноту. Щоб повторно не витрачати ресурси на вже виконану роботу, необхідно відсіювати такі дублікати якомога раніше. Для цього після кожного розширення кліки перевіряється, чи не стала вона занадто схожа на одну з вже завершених клік. Як метрика використовується функція відстані між спільнотами:

$$\delta_E(S, S') = 1 - \frac{|S \cap S'|}{\min(|S|, |S'|)}, \tag{3.10}$$

відсоток вершин меншої спільноти, які не включені в більшу спільноту. Тоді потенційними дублікатами S будуть спільноти, що знаходяться на відстані не далі ε від нього.

Для порівняння розбиття графа запропоновано чимала кількість різний заходів, але більшість з них підходить для роботи лише з непересічними розбивками. У даній роботі для оцінки описаних вище методів використовується міра Normalized Mutual Information (NMI) [22], яка підходить для оцінки пересічних розбиттsd (покриттів):

$$NMI(X,Y) = \frac{2(H(X) + H(Y) - H(X,Y))}{\mathrm{H(X)} + \mathrm{H(Y)}}, \quad (3.11)$$

де X і Y - розбиття графа на пересічні компоненти, H - ентропія Шеннона.

В якості тестових графів з пересічними компонентами були використані графи, згенеровані модифікованим алгоритмом LFR [23]. На вхід генератора передаються такі параметри:

- N - кількість вершин;
- k - середнє значення ступеня вершини;
- k_{max} - максимальне значення ступеня вершини;
- C_{min} - мінімальну кількість вершин в співтоваристві;
- C_{max} - максимальну кількість вершин в співтоваристві;
- τ_1 - експонента степенного розподілу ступеня вершини;
- τ_2 - експонента степенного розподілу розміру кластера;
- μ - параметр вираженості кластерної структури;
- on - кількість вершин, що належать більш ніж однієї спільноти;
- om - скільком спільнотам належить кожна вершина з області перетину.

3.5 Результати порівняльного аналізу алгоритмів

Для оцінки продуктивності алгоритмів на великих даних був підготовлений набір LFR-графів, які розрізняються кількістю вершин - від 100 до 1000000. Параметри LFR-графів представлені у таблиці 3.1. У процесі тестування при збільшенні кількості вершин у графі спостерігався експоненціальне зростання часу обробки.

Таблиця 3.1 – Параметри LFR-графів

Параметр	Графік a	Графік b
N	2000	2000
k	18-90	18-36
k_{max}	120	120
C_{min}	60	60
C_{max}	100	100
τ_1	2	2
τ_2	1	1
μ	0.2	0.2
on	2000	0-2000
om	1-5	2

На рисунку 3.6 представлені графіки падіння продуктивності в залежності від кількості вершин у графі.

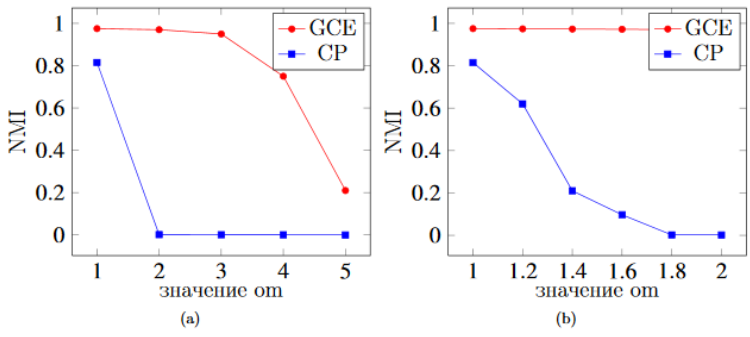

Рисунок 3.6 – Значення показника NMI для різних значень om

Висновок до розділу

У розділі «Математичне забезпечення» було сформульовано змістовну та математичну постановки задачі порівняльного аналізу методів пошуку прихованих підгруп у соціальних мережах.

Було розглянуто деякі методи з рідних сімейств та обрані методи, які будуть реалізовані в програмному продукті. Методом пошуку підгруп було обрано локальні методи, а саме перколяція клік (k-Clique percolation) та жадібний алгоритм розширення клік (Greedy Clique Expansion). Також було наведено детальний опис алгоритмів розв’язання поставленої задачі.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для програмного застосування я обрала мову програмування Java, та СУБД SQLite, далі ми розглянемо їх переваги та недоліки і порівняємо з аналогом.

Java, спочатку випущений Sun Microsystems в 1995 році, є мовою програмування загального призначення, який був розроблений з конкретною метою, що дозволяє розробникам "write once, run anywhere", тобто написати код один раз і запускати в будь-якому місці. Java-додатка скомпільовані в байт-код, який може запускатися при реалізації віртуальної машини Java (JVM). Подібно CLI, JVM допомагає подолати розрив між вихідним кодом і 1 і 0, які розуміє комп'ютер.

C # - це мова програмування загального призначення, який вперше з'явився в 2000 році в рамках ініціативи Microsoft .NET. Він був розроблений для загальної мовної інфраструктури (CLI) - відкритої специфікації, розробленої Microsoft і стандартизованої ISO і ECMA. Додатки C # скомпільовані в байт-код, який може запускатися при реалізації CLI.

Поява як Java, так і C #, тісно пов'язане з переходом від низькорівневих мов програмування, таких як мови програмування C ++, до мов вищого рівня, які компілюються в байт-код. Байт-код можна запустити на віртуальній машині. З цим пов'язаний ряд переваг, в першу чергу, можливість написання коду, який буде зрозумілий людині і буде працювати на будь-якій апаратній архітектурі, на якій встановлена віртуальна машина. Якщо відкинути синтаксичні примхи в сторону, то не дивно, що ці два подібні між собою мови так популярні для розробників додатків. Ось кілька основних подібностей між C # і Java:

– Безпека типів. Помилка типу виникає, коли тип даних одного об'єкта помилково призначається іншому об'єкту, створюючи ненавмисні

побічні ефекти. І C #, і Java працюють на те, щоб гарантувати виявлення таких типів незаконних привидів під час компіляції. Якщо приведення не може бути застосоване до нового типу, тоді під час виконання такі винятки будуть видалені.

– Прибирання сміття: На мовах нижчого рівня управління пам'яттю може бути стомлюючим, адже потрібно пам'ятати про те, що необхідно правильно видалити нові об'єкти, щоб звільнити ресурси. На C # і Java є вбудована прибирання сміття, яка допомагає запобігти витоку пам'яті шляхом видалення об'єктів, які більше не використовуються додатком. Витоку пам'яті все ще можуть виникати, але завдяки основам управління пам'яттю - це вже не ваша проблема.
– Одиночне спадкоємство. Обидві мови підтримують одиночне спадкоємство - це означає, що існує тільки один шлях з будь-якого базового класу в будь-який з його похідних класів. Це обмежує ненавмисні побічні ефекти, які можуть виникати при наявності декількох шляхів між декількома базовими класами і похідними класами. Diamond pattern - книжковий приклад цієї проблеми.
– Інтерфейси. Інтерфейс являє собою абстрактний клас, де всі методи абстрактні. Абстрактним методом є той метод, який оголошений, але не містить подробиць його реалізації. Код, що визначає будь-які методи або властивості, певні інтерфейсом, має надаватися класом, який його реалізує. Це допомагає уникнути двозначності паттерна diamond, оскільки завжди ясно, який базовий клас реалізує даний похідний клас під час виконання. Результатом є чиста ієрархія лінійних класів одиночного наслідування в поєднанні з деякою універсальністю множинного спадкоємства. Фактично використання абстрактних класів є одним із способів множинного успадкування мов, які можуть подолати проблему паттерна diamond.

Важливо пам'ятати, що C # бере свій початок в бажанні Microsoft мати власний «Java-подібний» мову для платформи .NET. Оскільки C # не створювалася в вакуумі, нові функції були додані і налаштовані для вирішення проблем, з якими стикалися розробники Microsoft, коли вони спочатку намагалися створити свою платформу на Visual J ++. У той же час співтовариство Java з відкритим вихідним кодом продовжувала зростати і між цими двома мовами розвивалася гонка технічних озброєнь. Ось деякі з основних відмінностей між C # і Java.

– Windows vs open-source. Хоча існують реалізації з відкритим вихідним кодом, C # в основному використовується в розробці для платформ Microsoft - .NET Framework CLR і є найбільш широко використовуваної реалізацією CLI. На іншому кінці спектру Java має величезну екосистему з відкритим вихідним кодом і у нього відкрилося друге дихання частково завдяки тому, що Google використовує JVM для Android.
– Підтримка узагальнень (Generics): Generics покращує перевірку типів за допомогою компілятора, в основному видаляючи приведення з вихідного коду. В Java кошти узагальнень реалізуються з використанням стирань. Параметри загального типу «стираються», а при компіляції в байт-код додаються приведення. C # також використовує узагальнення, інтегруючи його в CLI і надаючи інформацію про тип під час виконання, що дає невелике збільшення продуктивності.
– Підтримка делегатів (покажчиків): У C # є делегати, які по суті є як методів, які можуть бути викликані не повідомляючи цільового об'єкта. Для досягнення такої ж функціональності в Java вам необхідно використовувати інтерфейс з одним методом або іншим способом обходу, який може зажадати нетривіального кількості додаткового коду, в залежності від програми.

– Перевіряються виключення: Java розрізняє два типи винятків - Перевіряються і непроверяемие. C # вибрав більш мінімалістський підхід, маючи тільки один тип винятку. Хоча здатність ловити виключення може бути корисна, вона також може мати негативний вплив на масштабованість і контроль версій.
– Поліморфізм: C # і Java використовують дуже різні підходи до поліморфізму. Java допускає поліморфізм за замовчуванням, C # же повинен викликати ключове слово «virtual» в базовому класі і ключове слово «override» в похідному класі.
– Перерахування (Enums): в C # перерахування представляють собою прості списки іменованих констант, де базовий тип повинен бути цілим. Java являє перерахування більш глибоко, розглядаючи його як іменований екземпляр типу, що спрощує додавання користувацького поведінки окремих перерахуванням [20].

Реляційні системи реалізують реляційну модель роботи з даними, яка визначає всю збережену інформацію як набір пов'язаних записів і атрибутів в таблиці.

СУБД такого типу використовують структури (таблиці) для зберігання і роботи з даними. Кожен стовпець (атрибут) містить свій тип інформації. Кожен запис в базі даних, що володіє унікальним ключем, передається в рядок таблиці, і її атрибути відображаються в стовпцях таблиці.

SQLite - це дивовижна бібліотека, вбудована в додаток, яке її використовує. Будучи файлової БД, вона надає відмінний набір інструментів для більш простої (в порівнянні з серверними БД) обробки будь-яких видів даних.

Коли додаток використовує SQLite, їх зв'язок проводиться за допомогою функціональних і прямих викликів файлів, що містять дані (наприклад, баз даних SQLite), а не якогось інтерфейсу, що підвищує швидкість і продуктивність операцій.

Переваги:

- Файлова: вся база даних зберігається в одному файлі, що полегшує переміщення.
- Стандартизована: SQLite використовує SQL; деякі функції опущені (RIGHT OUTER JOIN або FOR EACH STATEMENT), однак, є і деякі нові.
- Дуже добре підходить для розробки і навіть тестування: під час етапу розробки більшості потрібно масштабується,. SQLite, зі своїм багатим набором функцій, може надати більш ніж достатній функціонал, при цьому будучи досить простий для роботи з одним файлом і пов'язаної сішной бібліотекою.

Недоліки:

- Відсутність призначеного для користувача управління: просунуті БД надають користувачам можливість управляти зв'язками в таблицях відповідно до привілеями, але у SQLite такої функції немає.
- Неможливість додаткової настройки: знову-таки, SQLite не можна зробити більш продуктивною, подлубавшись в настройках - так вже вона влаштована.

MySQL - це найпопулярніша з усіх великих серверних БД. Розібратися в ній дуже просто, та й в мережі про неї можна знайти велику кількість інформації. Хоча MySQL і не намагається повністю реалізувати SQL-стандарти, вона пропонує широкий функціонал. Додатки спілкуються з базою даних через процес-демон.

Переваги:

- Простота: MySQL легко встановлюється. Існує багато сторонніх інструментів, включаючи візуальні, що полегшують початок роботи з БД.

- Багато функцій: MySQL підтримує більшу частину функціоналу SQL.
- Безпека: в MySQL вбудовано багато функцій безпеки.
- Потужність і масштабованість: MySQL може працювати з дійсно великими обсягами даних, і непогано підходить для масштабованих додатків.
- Швидкість: нехтування деякими стандартами дозволяє MySQL працювати продуктивніше, місцями зрізуючи на поворотах.

Недоліки:

- Відомі обмеження: по визначенню, MySQL не може зробити все, що завгодно, і в ній присутні певні обмеження функціональності.
- Питання надійності: деякі операції реалізовані менш надійно, ніж в інших РСУБД.
- Застій в розробці: хоча MySQL і є open-source продуктом, робота над нею сильно загальмована. Проте, існує кілька БД, повністю заснованих на MySQL (наприклад, MariaDB) [21].

Отже виходячи з порівнянь засобів розробки визначимо, що для нашої задачі більш ніж достатньо використовувати СУБД SQLite, через те, що вона дуже компактно, та гарно працює.

Основною мовою програмування була вибрана мова Java. Вона є високорівневою і має досить високу продуктивність.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Для правильної роботи розробленого продукту до складу технічних засобів мають входити:

- комп’ютер з такою конфігурацією:
  1) процесор з тактовою частотою не нижче 1 ГГц;
  2) достатній об’єм оперативної пам’яті (не менше 256 МБ);

3) інші складові можуть мати будь-які параметри, тому що вони не значним чином впливають на роботу програми;

– додатково має бути встановлене таке програмне забезпечення:
  1) операційна система Windows 7 і вище;
  2) база даних Oracle Database 10g Express Edition;
  3) Net Framework 3.5 і вище;
– комп’ютерна периферія, до складу якої входить:
  1) монітор;
  2) мишка;
  3) клавіатура.

4.3 Архітектура програмного забезпечення

Доцільна побудова наступних діаграм UML фази уточнення(таблиця 4.1):

Таблиця 4.1 – Перелік потрібних діаграм

Тип діаграми	Роль у процесі проектування
Діаграма послідовності	У першому наближенні визначає набір класів та їхню послідовність взаємодії для вирішення певної задачі
Діаграма класів	Визначає набір класів та їхню взаємодію задля вирішення певної задачі (виконання процесу)
Діаграма компонентів	Визначає групування класів у компоненти (а затим у виконувані або бібліотечні двійкові файли) та ланцюжки виклику компонентів іншими компонентами під час виконання програми.

4.3.1 Діаграма класів

Діаграма класів призначена для надання статичної структури моделі системи в термінології класів об’єктно-орієнтованого програмування. Діаграма класів відображує різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти й підсистеми, а також описує їхню внутрішню структуру й типи відносин. На даній діаграмі не вказується інформація про часові аспекти функціонування системи.

Діаграма класів (class diagram**)** - діаграма на якій представлена сукупність декларативних або статичних елементів моделі, таких як класи з атрибутами й операціями, а також відношення, що їх з’єднують.

Діаграму класів наведено в частині графічного матеріалу[18].

4.3.2 Діаграма послідовності

Діаграма послідовностей відображає взаємодію об'єктів в динаміці. Головними елементами діаграм послідовності є об’єкти, які є логічними сутностями, що представляють окремі елементи системи та повідомлення, якими вони обмінюються.

Діаграма послідовностей відноситься до діаграм взаємодії UML, що описує поведінкові аспекти системи, але розглядає взаємодію об'єктів в часі. Іншими словами, діаграма послідовностей відображає часові особливості передачі і прийому повідомлень об'єктами.

Процес застосування алгоритмів пошуку підгруп у соціальних мережах на реальних даних відбувається наступним чином. Користувач запускає програму та обирає один з двох алгоритмів пошуку підгруп.

Об’єкти інтерфейсу взаємодіють з користувачем, відображаючи форму «Застосування алгоритмів на реальних даних», де користувач має змогу обрати алгоритм.

Об’єкти бізнес-логіки виконують запити до бази даних. Об’єкти бізнес логіки відповідають за основний функціонал застосування.

База даних надає дані для їх подальшої обробки об’єктами бізнес-логіки.

Потім об’єкти інтерфейсу відображають дані, які надійшли з бази даних та були оброблені об’єктами бізнес-логіки.

Діаграму послідовності наведено в частині графічного матеріалу[19].

4.3.3 Діаграма компонентів

Діаграма компонентів належить до категорії структурних діаграм.

Діаграми компонентів - це один з двох видів діаграм, застосовуваних при моделюванні фізичних аспектів об'єктно-орієнтованої системи. Вони показують організацію наборів компонентів і залежності між ними. Діаграми компонентів застосовуються для моделювання статичного виду системи з точки зору реалізації.

Діаграми компонентів важливі не тільки для візуалізації, специфікування і документування системи, заснованої на компонентах, але і для створення виконуваних систем шляхом прямого і зворотнього проектування.

Як можна побачити на діаграмі, застосування має один інтерфейс, який надсилає запити до бази даних з метою виконати маніпуляцію або відобразити певну інформацію.

Діаграму компонентів наведено в частині графічного матеріалу.

4.3.4 Специфікація функцій

Для реалізації застосунку порівняльного аналізу методів пошуку прихованих підгруп у соціальній мережі було розроблено методи для кожного класу.

В таблиці 4.1 наведено назви основних функцій та дії, що вони виконують.

Таблиця 4.2 – Опис функцій

Назва функції	**Дія**
read_data()	Зчитує дані користувачів з БД
write_data()	Записує дані користувачів до БД
findConnectedComponents()	Знаходить всі зв’язані вершини графу
getKCliquesFromGraph()	Знаходить всі максимальні кліки і повертає їх у вигляді множин. Реалізує алгоритм Брон-Кербош
end()	Повідомляє нам, якщо вузол усередині вже знайдений і підключений до всіх вузлів-кандидатів
max_clique()	Пошук максимальної кліки
countFitness()	Розраховує для кожної спільноти приріст значення адекватності при додаванні однієї з вершин фронту
find-MaxFitness()	Знаходить максимум серед усіх значень і додає відповідну вершину в спільноту
getAdjacen()	Визначаються групи пересічних спільнот щодо деяких точок перетину
rejectDuplicated()	Виявляються незавершені спільноти, які слабо відрізняються від суміжних з ними завершених. Такі спільноти видаляються
result()	Формування результатів роботи алгоритмів
count_time()	Підрахунок часу роботи кожного алгоритму

Продовження таблиці 4.2

count_quality()	Підрахунок якості роботи кожного алгоритму
build()	Зберігання значень для побудови порівняльних алгоритмів
generate_data()	Генерація набору LFR – графів для порівняльного аналізу

Висновок до розділу

У розділі «Програмне та технічне забезпечення» проведено порівняння деяких існуючих засобів розробки. Після аналізу, обрані засоби розробки програмного продукту, для порівняльного аналізу методів пошуку підгруп у соціальних мережах.

Наведені вимоги до технічного забезпечення, які необхідні для коректного функціонування програмного продукту.

Представлена модель архітектури програмного забезпечення.

Наведено детальний опис діаграми класів, що демонструє наявні в застосуванні класи, опис діаграми послідовності, яка демонструє порядок взаємодії об’єктів із базою даних та користувачем, опис діаграми компонентів, яка показує розбиття програмної частини застосування на структурні компоненти та їх залежність один з одним.

Наведена специфікація функцій, які використовуються у розробці програмного продукту.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Для того, щоб працювати з системою, користувач повинен мати мінімальні навички роботи з комп’ютером.

Робота з програмним продуктом починається з головної форми. З неї є можливість здійснити перехід на форму, де демонструється робота алгоритму на реальній мережі, та на форма, де здійснюється порівняльний аналіз двох алгоритмів. Також на головній формі є кнопка, за допомогою якої, користувач може повністю вийти з системи.

На рисунку 5.1 наведено головну форму.

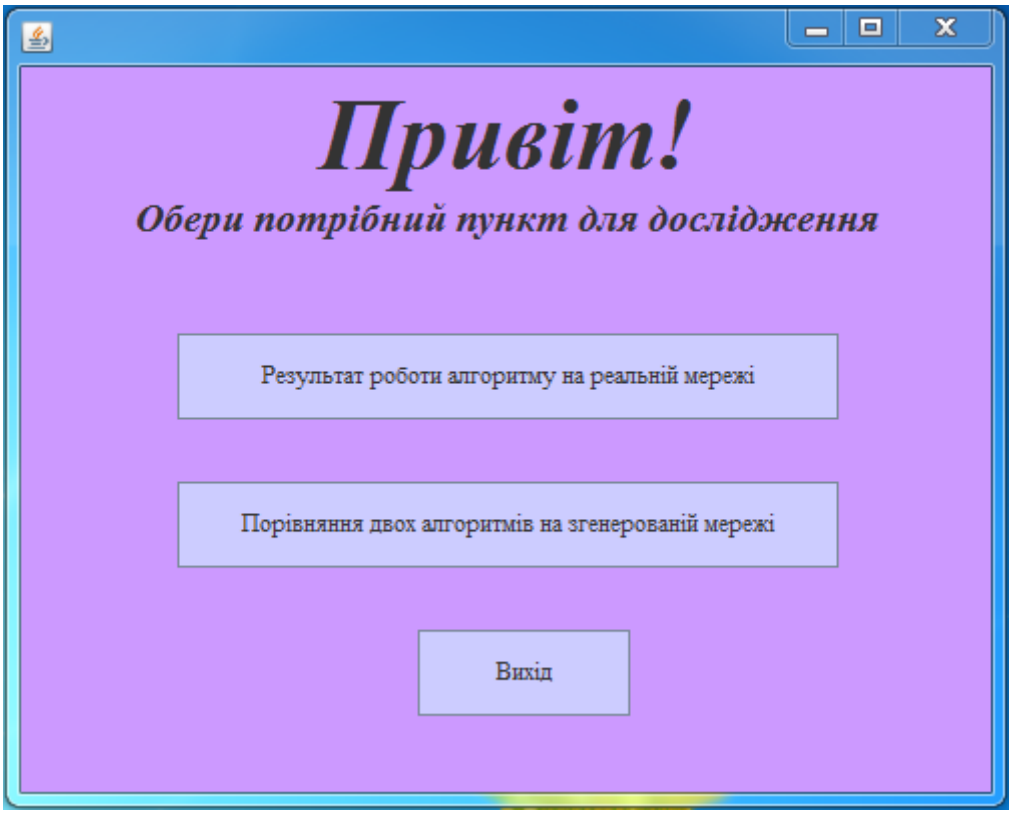

Рисунок 5.1 - Головна форма

Натисніть на кнопку «Результат роботи алгоритму на реальній мережі». Відкриється форма, де демонструється робота обраного алгоритму (рисунок 5.2).

Рисунок 5.2 – Форма для демонстрації роботи алгоритмів

Далі потрібно обрати один алгоритм та натиснути кнопку «Почати пошук». Після цих дій ми можемо бачити результат роботи обраного алгоритму у вигляді таблиці особових даних користувачів соціальної мережі (рисунок 5.3).

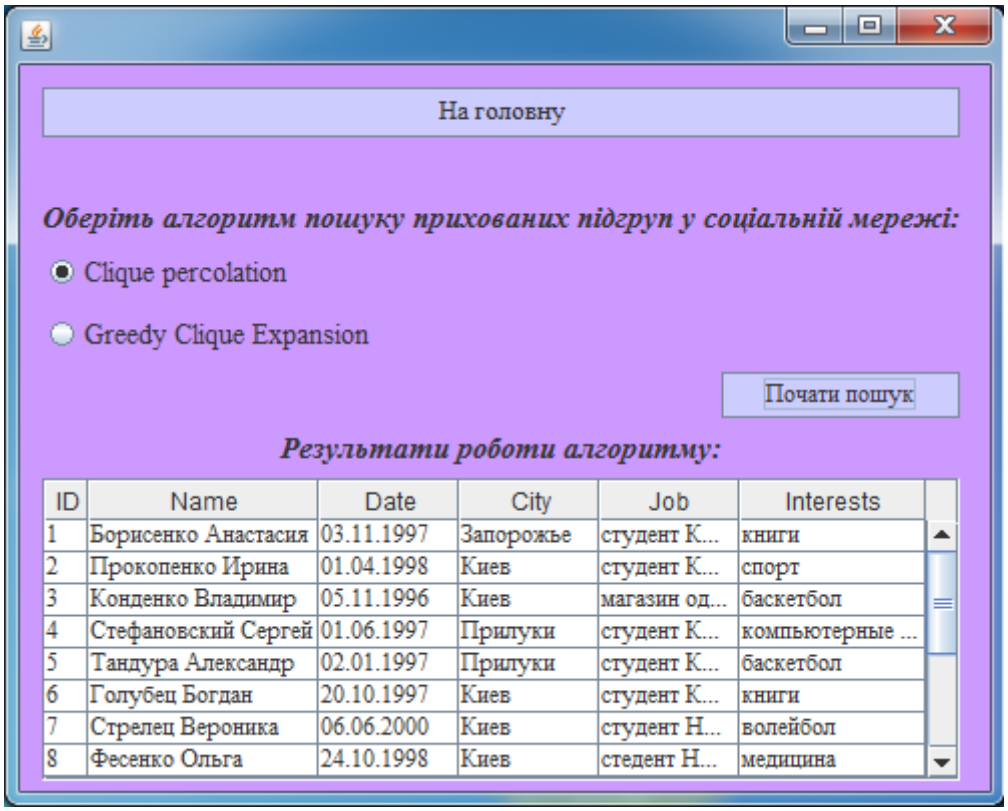

Рисунок 5.3 – Форма з результатом роботи алгоритму

Щоб повернутись на головну форму треба натиснути кнопку «На головну». Після переходу, натиснемо кнопку «Порівняння двох алгоритмів на згенерованій мережі» і перейдемо на форму, де буду здійснюватись порівняння (рисунок 5.4).

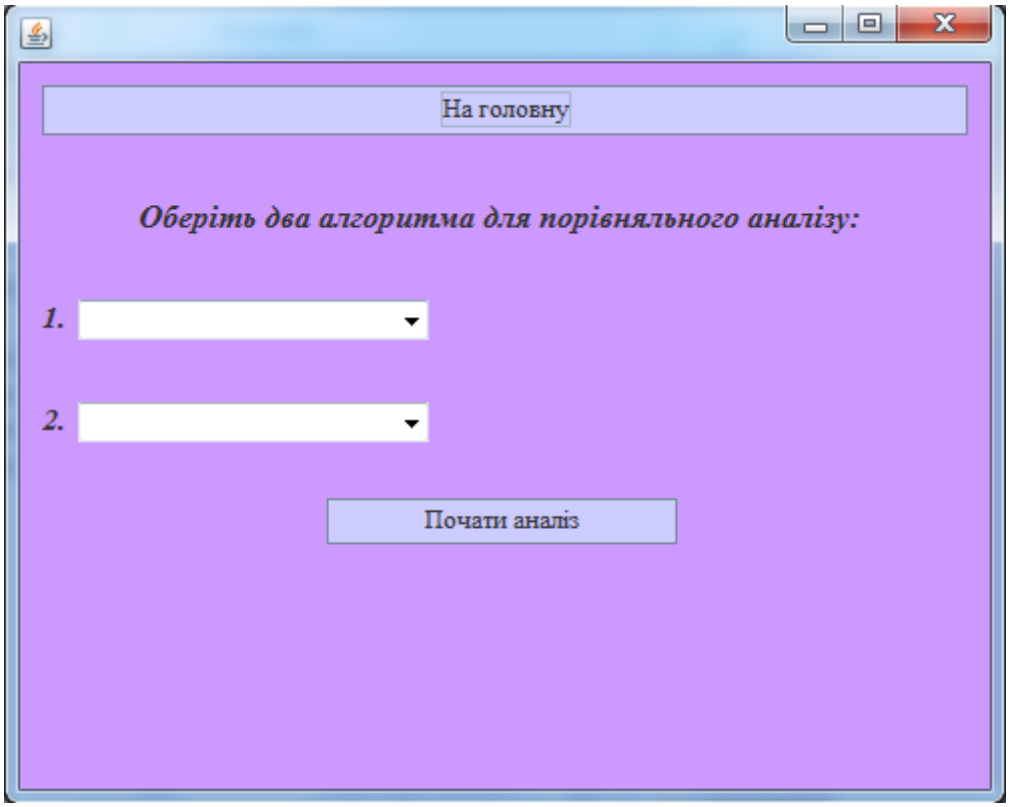

Рисунок 5.4 – Форма, де здійснюється порівняння

Щоб здійснити порівняння двох алгоритмів треба обрати два різних алгоритма з випадаючого списку існуючих алгоритмів та натиснути кнопку «Почати аналіз» (рисунок 5.5).

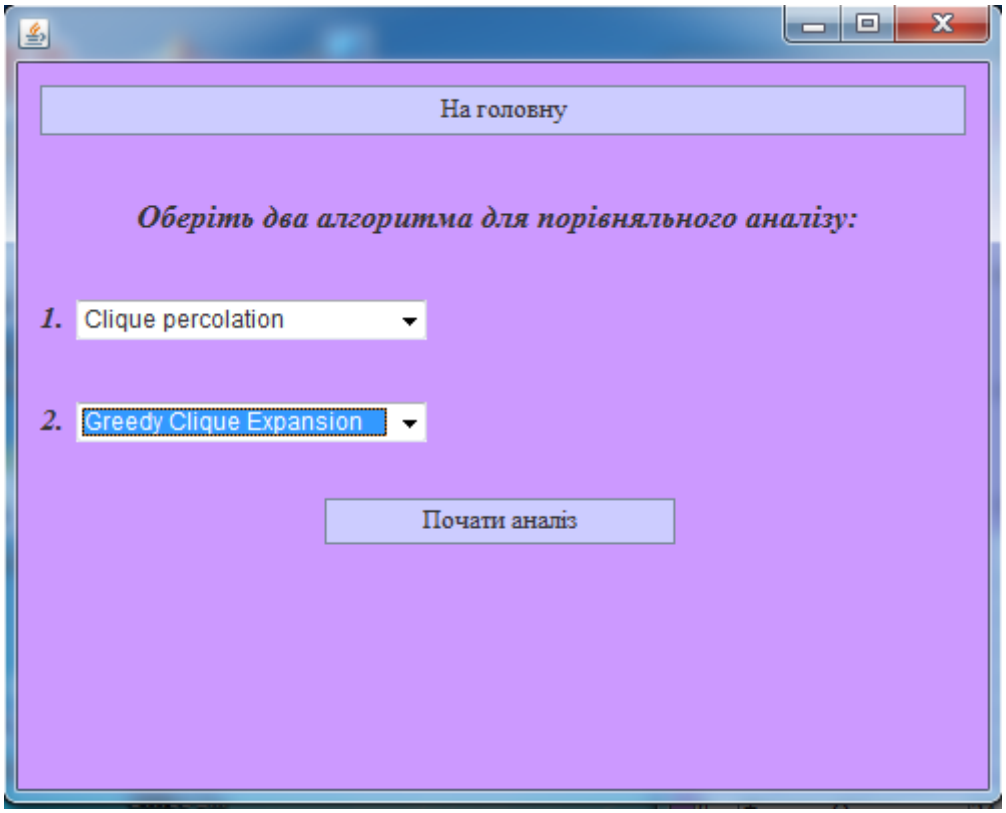

Рисунок 5.5 – Форма з даними для початку порівняння

Після закінчення порівняльного аналізу, користувач може побачити порівняльні графіки двох алгоритмів за різними критеріями та на різній кількості користувачем соціальної мережі. Щоб повернутись до головної форми треба натиснути кнопку «На головну». Для того щоб вийти з системи треба на головній формі натиснути кнопку «Вихід».

5.2 Випробування програмного продукту

В цьому підрозділі наведено опис тестів і порядок їх виконання для перевірки відповідності програмного забезпечення комплексу задач функціональним вимогам, представленим у технічному завданні на створення застосування для порівняльного аналізу методів виявлення прихованих підгруп у соціальних мережах..

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій

комплексу задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

В процесі тестування були перевірена уся функціональність комплексу задач (КЗ). У наступних таблицях наведений перелік випробувань основних функціональних можливостей (табл. 5.1 – 5.2).

Таблиця 5.1 – Вибір алгоритму

Мета тесту:	Перевірка функції «Перевірка вибору алгоритму»
Початковий стан КЗ:	Відкрита форма «Застосування алгоритмів на реальних даних»
Вхідні данні:	Один з двох алгоритмів
Схема проведення тесту:	Обрати алгоритм. Натиснути кнопку «Почати»
Очікуваний результат:	Вікно повідомлення про помилку вводу. Перехід на форму «Результат роботи» не виконано «Відсутній алгоритму пошуку»
Стан КЗ після проведення випробувань:	Відкрита форма з результатами досліджень

Таблиця 5.2 – Результат досліджень

Мета тесту:	Перевірка функції «Пошук клік»
Початковий стан КЗ:	Відкрита форма «Порівняння двох алгоритмів»
Вхідні данні:	Набір згенерованих пар вершин
Схема проведення тесту:	Натиснути кнопку «Почати»
Очікуваний результат:	Вікно повідомлення про помилку. Дію не виконано «Кліки відсутні»
Стан КЗ після проведення випробувань:	Оновлена форма з порівняльними графіками

Висновок до розділу

У «Технологічному розділі» була описана інструкція користувача, яка демонструє існуючі можливості програмного продукту за допомогою детального опису та скріншотів.

Детально описано, як користувачу працювати із програмним продуктом для дослідження методів пошуку.

Наведені мета випробовувань, загальні положення та результати випробувань, а саме розроблені та успішно пройдені тестові сценарії для програмного продукту.

ЗАГАЛЬНІ ВИСНОВКИ

У дипломній роботі були розглянуті та порівняні методи виявлення прихованих підгруп у соціальних мережах.

Розділ загальних положень надає інформацію про предметне середовище роботи. Описані діючі актори системи (користувач) та їх функції. Проведено аналіз аналогів застосування.

Було наведено опис інформаційного забезпечення до програмного продукту , що детально описує структуру бази даних, вхідні та вихідні дані. Структура бази даних описана в таблицях, які містять у собі інформацію про унікальні та обов’язкові поля сутностей, виділені зовнішні і первинні ключі бази даних.

У розділі програмної та технічної підтримки описані технології, які використані в програмному продукті. Обґрунтовуються технології та їх особливості, які використовуються в даній розробці. В якості БД була використана SQLite, а інструментом розробки вибрана мова програмування Java. Наведені мінімальні вимоги до технічного забезпечення. Розглядаються описи класів, на яких базується архітектура розробки.

У технологічному розділі було наведено керівництво користувача в якому описуються детальні інструкції до розробленого застосування. Описані методи випробування на основі тестових сценаріїв.

Дане застосування автоматизує процес дослідження, для вибору найефективнішого алгоритму пошуку прихованих підгруп у соціальних мережах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wasserman, S. Social network analysis / S. Wasserman, K. Faust. 1994
2. Girvan, M. Community structure in social and biological networks /M. Girvan, M. Newman // Proceedings of the National Academy ofSciences. 2002. V. 99, No 12. P. 7821–7826
3. Rives, A. Modular organization of cellular networks / A. Rives,T. Galitski // Proceedings of the National Academy of Sciences. 2003. V. 100, No 3. P. 1128–1133.
4. Chen, J. Detecting functional modules in the yeast protein–proteininteraction network / J. Chen, B. Yuan // Bioinformatics. 2006. V. 22, No 18. P. 2283–2290.
5. Krishnamurthy, B. On network-aware clustering of web clients /B. Krishnamurthy, J. Wang // ACM SIGCOMM ComputerCommunication Review / ACM. V. 30. 2000. P. 97–110
6. Agrawal, R. Algorithms for searching massive graphs / R. Agrawal,H. Jagadish // Knowledge and Data Engineering, IEEE Transactions on. 1994. V. 6, No 2. P. 225–238
7. Steenstrup, M. Cluster-based networks / M. Steenstrup // Ad hocnetworking / Addison-Wesley Longman Publishing Co., Inc. 2001. P. 75–138.
8. Burt, R. Positions in networks / R. Burt // Social forces. 1976. V. 55, No 1. P. 93–12
9. Freeman, L. A set of measures of centrality based on betweenness /L. Freeman // Sociometry. 1977. P. 35
10. Girvan, M. Community structure in social and biological networks /M. Girvan, M. Newman // Proceedings of the National Academy ofSciences. 2002. V. 99, No 12. P. 7821–7826
11. Newman, M. Finding and evaluating community structure in networks /M. Newman, M. Girvan // Physical review E. 2004. V. 69, No 2. P. 026113.

12. Evans, T. Line graphs of weighted networks for overlapping communities T. Evans, R. Lambiotte // The European Physical Journal B-CondensedMatter and Complex Systems. 2010. V. 77, No 2. P. 265–272
13. Detecting highly overlapping community structure by greedy cliqueexpansion / C. Lee, F. Reid, A. McDaid, N. Hurley // Arxiv preprintarXiv:1002.1827. 2010
14. Uncovering the overlapping community structure of complex networks innature and society / G. Palla, I. Der´enyi, I. Farkas, T. Vicsek // Nature. 2005. V. 435, No 7043. P. 814–818
15. Sequential algorithm for fast clique percolation / J. Kumpula, M. Kivel □a,K. Kaski, J. Saram □aki // Physical Review E. 2008. V. 78, No 2. P. 026109.
16. Bron, C. Algorithm 457: finding all cliques of an undirected graph /C. Bron, J. Kerbosch // Communications of the ACM. 1973. V. 16,No 9. P. 575–577.
17. Tomita, E. The worst-case time complexity for generating all maximalcliques and computational experiments / E. Tomita, A. Tanaka,H. Takahashi // Theoretical Computer Science. 2006. V. 363,No 1. P. 28–42
18. UML - уніфікована мова моделювання. веб-сайт. URL: http://dl.my-konotop.org/mod/page/view.php?id=237
19. Опис взаємодії за допомогою Sequencediagram. веб-сайт. URL: http://www.tsatu.edu.ua/kn/wp-content/uploads/sites/16/laboratorna-robota-9-diahrama-poslidovnosti.pdf
20. C# против Java: какой язык программирования общего назначения выбрать? веб-сайт. URL: https://itvdn.com/ru/blog/article/csharp_vs_java
21. SQLite, MySQL и PostgreSQL: сравниваем популярные реляционные СУБД. веб-сайт. URL: https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/

22. Lancichinetti, A. Detecting the overlapping and hierarchical communitystructure in complex networks / A. Lancichinetti, S. Fortunato,J. Kert´esz // New Journal of Physics. – 2009. – V. 11. – P. 033015.
23. Lancichinetti, A. Benchmarks for testing community detection algorithmson directed and weighted graphs with overlapping communities /A. Lancichinetti, S. Fortunato // Physical Review E. – 2009. – V. 80,No 1. – P. 01611.

Додаток А

Тексти програмного коду

Комплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі

(Найменування програми (документа))

DVD-R

(Вид носія даних)

14 арк, 244 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2019 року

```
import com.google.common.collect.Sets;

import java.util.*;

/**
 * @param <T> The graph's nodes type.
 */
public class CliquePercolation<T> {

    /**
     * Clique size
     */
    private int k;

    /**
     * Threshold used to "binarize" the weights between nodes
     */
    private double threshold;


    /**
     * @param k {@link CliquePercolation#k}
     */
    public CliquePercolation(int k, double threshold) {
        if (k < 2) {
            throw new IllegalArgumentException("k must be greater than 1.");
        }

        if (threshold <= 0 || threshold >= 1) {
            throw new IllegalArgumentException("threshold must be strictly between 0 and 1.");
        }

        this.k = k;
        this.threshold = threshold;
    }

    /**
     * Implementation of Clique Percolation using an alternative class for graphs.
     */
    public Solucio<T> generateSolution(AffinitiesGraph<T> g) {
        return generateSolution(g, true);
    }

    /**
     * Implementation of Clique Percolation using an alternative class for graphs.
     */
    public Solucio<T> generateSolution(AffinitiesGraph<T> g, boolean processLeftOutNodes) {
        long startTime = System.currentTimeMillis();
```

```
        // This is the complicated part: obtaining the Cliques.
        ArrayList<HashSet<T>> kCliques = getKCliquesFromGraph(g);

        // Now we are creating a graph where the nodes are the previously found cliques.
        AffinitiesGraph<HashSet<T>> kCliqueGraf = new AffinitiesGraph<>(kCliques.size());
        kCliqueGraf.addNodes(kCliques);

        for (int i=0; i<kCliques.size(); i++) {
           for (int j=i+1; j<kCliques.size(); j++) {

              if (Sets.intersection(kCliques.get(i), kCliques.get(j)).size() >= k-1) {
                 kCliqueGraf.addEdge(kCliques.get(i), kCliques.get(j), 1.0);
              }
           }
        }

        // Now we are finding the "components" of the clique-graph
        ArrayList<HashSet<T>> communities = findConnectedComponents(kCliqueGraf);

        // Now we apply "Maity-Kumar tweaks" to "classify" the left out nodes.
        if (processLeftOutNodes) {
           applyMaityKumarTweaks(g, communities);
        }

        // Now we "translate" what we've found to a Solution instance.
        Solucio<T> solution = new Solucio<>();

        int communityId = 0;
        for (HashSet<T> community : communities) {
           solution.afegirComunitat(communityId);

           for(T node : community) {
              solution.afegirNode(node, communityId);
           }

           communityId++;
        }

        solution.setTempsExec((double)System.currentTimeMillis()-startTime);
        return solution;
     }

     private ArrayList<HashSet<T>> findConnectedComponents(AffinitiesGraph<HashSet<T>>
kCliqueGraf) {
        ArrayList<HashSet<T>> communities = new ArrayList<>();
        HashSet<HashSet<T>> processedCliques = new HashSet<>();

        for(HashSet<T> seedClique : kCliqueGraf) {
           if (processedCliques.contains(seedClique)) continue;

           Queue<HashSet<T>> cliquesQueue = new ArrayDeque<>();
```

```
            cliquesQueue.add(seedClique);

            HashSet<T> community = new HashSet<>(seedClique.size());

            while (cliquesQueue.size() > 0) {
               HashSet<T> tmpClique = cliquesQueue.poll();

               // Making grow the community
               community.addAll(tmpClique);
               processedCliques.add(tmpClique);

               for (HashSet<T> stemClique : kCliqueGraf) {
                  if (processedCliques.contains(stemClique)) continue;

                  if (kCliqueGraf.hasEdge(seedClique, stemClique)) {
                     cliquesQueue.add(stemClique);
                  }
               }
            }

            communities.add(community);
         }

         return communities;
      }

      private void applyMaityKumarTweaks(AffinitiesGraph<T> g, ArrayList<HashSet<T>>
communities) {
         HashSet<T> leftOutNodes = g.getNodes();
         for (HashSet<T> community : communities) {
            leftOutNodes.removeAll(community);
         }

         // We take apart the "very lonely" nodes because Maity-Kumar tweaks don't handle them in
a graceful way.
         // We'll create a singleton community for every one of them at the end of the Maity-Kumar
tweaks.
         HashSet<T> singletons = new HashSet<>();

         for(T leftOutNode : leftOutNodes) {
            boolean connected = false;
            for (T node : g) {
               if (node.equals(leftOutNode)) continue;

               if (g.getEdge(leftOutNode, node) >= threshold)  {
                  connected = true;
                  break;
               }
            }
            if (!connected) {
               singletons.add(leftOutNode);
```

```
            }
        }
        leftOutNodes.removeAll(singletons);

        // This numbers will be used to compute a "belonging score".
        HashMap<T, Double> weightsSums = new HashMap<>(leftOutNodes.size());
        for (T leftOutNode : leftOutNodes) {
            double sum = 0;

            for (T node : g) {
                if (node.equals(leftOutNode)) continue;

                sum += g.getEdge(leftOutNode, node);
            }

            weightsSums.put(leftOutNode, sum);
        }

        // The nodes aren't added to existent communities at the same time because we want to
optimize the "classification".
        while (leftOutNodes.size() > 0) {
            double maxBelongingScore = 0;
            ImmutablePair<T, HashSet<T>> nextAnnexation = null;

            for (T leftOutNode : leftOutNodes) {
                for (HashSet<T> community : communities) {
                    double belongingScore = 0;

                    for (T node : community) {
                        if (node.equals(leftOutNode)) continue;
                        belongingScore += g.getEdge(leftOutNode, node);
                    }

                    belongingScore /= weightsSums.get(leftOutNode);

                    if (belongingScore > maxBelongingScore) {
                        maxBelongingScore = belongingScore;
                        nextAnnexation = new ImmutablePair<>(leftOutNode, community);
                    }
                }
            }

            nextAnnexation.getSecond().add(nextAnnexation.getFirst());
            leftOutNodes.remove(nextAnnexation.getFirst());
        }

        // Finally, we add the singletons we previously segregated.
        for (T singleton : singletons) {
            HashSet<T> tmpSingleton = new HashSet<>(1);
            tmpSingleton.add(singleton);
            communities.add(tmpSingleton);
```

```
    }
}

/**
 * Finds all the maximal cliques of g, and returns it in form of sets.
 * Implements the Bron-Kerbosch algorithm.
 */
private ArrayList<HashSet<T>> getKCliquesFromGraph(AffinitiesGraph<T> g) {
    ArrayList<HashSet<T>> cliques = new ArrayList<>();
    ArrayList<T> potentialClique = new ArrayList<>();
    ArrayList<T> alreadyFound = new ArrayList<>();
    HashSet<T> candidates = g.getNodes();

    findCliques(potentialClique, candidates, alreadyFound, cliques, g);

    return cliques;
}

/**
 * Body of the Bron-Kerbosch algorithm.
 *
 * @param potentialClique
 * @param candidates
 * @param alreadyFound
 */
private void findCliques(
        ArrayList<T> potentialClique, Collection<T> candidates, ArrayList<T> alreadyFound,
        ArrayList<HashSet<T>> cliques, AffinitiesGraph<T> g
) {

    ArrayList<T> candidatesArray = new ArrayList<T>(candidates);

    if (!end(candidates, alreadyFound, g)) {
        for (T candidate : candidatesArray) {
            ArrayList<T> newCandidates = new ArrayList<T>();
            ArrayList<T> newAlreadyFound = new ArrayList<T>();

            // move candidate node to potentialClique
            potentialClique.add(candidate);
            candidates.remove(candidate);

            // create newCandidates by removing nodes in candidates not
            // connected to candidate node
            for (T newCandidate : candidates) {
                if (g.getEdge(candidate, newCandidate) >= threshold)  {
                    newCandidates.add(newCandidate);
                }
            }

            // create newAlreadyFound by removing nodes in alreadyFound
            // not connected to candidate node
```

```
            for (T newFound : alreadyFound) {
               if (g.getEdge(candidate, newFound) >= threshold) {
                  newAlreadyFound.add(newFound);
               }
            }

            // if new_candidates and new_already_found are empty
            if (newCandidates.isEmpty() && newAlreadyFound.isEmpty()) {
               // potential_clique is maximal_clique
               if (potentialClique.size() >= k) {
                  cliques.add(new HashSet<T>(potentialClique));
               }
            } else {
               // recursive call
               findCliques(potentialClique, newCandidates, newAlreadyFound, cliques, g);
            }

            // move candidate_node from potential_clique to already_found;
            alreadyFound.add(candidate);
            potentialClique.remove(candidate);
         }
      }
   }

   /**
    * Tells us if a node inside alreadyFound us connected to all candidate nodes.
    */
   private boolean end(Collection<T> candidates, ArrayList<T> alreadyFound,
AffinitiesGraph<T> g)
   {
      int edgeCounter;

      for (T found : alreadyFound) {
         edgeCounter = 0;

         for (T candidate : candidates) {
            if (g.getEdge(found, candidate) >= threshold) {
               edgeCounter++;
            }
         }

         if (edgeCounter == candidates.size()) {
            return true;
         }
      }

      return false;
   }
}
```

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

	private static BundleContext context;

	static BundleContext getContext() {
		return context;
	}

	/*
	 * (non-Javadoc)
	 * @see
org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
	 */
	public void start(BundleContext bundleContext) throws Exception {
		Activator.context = bundleContext;
	}

	/*
	 * (non-Javadoc)
	 * @see
org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
	 */
	public void stop(BundleContext bundleContext) throws Exception {
		Activator.context = null;
	}


import java.io.InputStream;

import ru.ispras.modis.NetBlox.exceptions.GraphMiningException;
import ru.ispras.modis.NetBlox.exceptions.PluginException;
import ru.ispras.modis.NetBlox.graphAlgorithms.graphMining.AGraphMiner;
import ru.ispras.modis.NetBlox.graphAlgorithms.graphMining.GraphOnDrive;
import ru.ispras.modis.NetBlox.graphAlgorithms.graphMining.MinerResults;
import ru.ispras.modis.NetBlox.graphAlgorithms.graphMining.SupplementaryData;
import ru.ispras.modis.NetBlox.scenario.GraphMiningParametersSet;

public class GCEMinerCallback extends AGraphMiner {
	@Override
	public MinerResults mine(GraphOnDrive graphOnDrive, SupplementaryData
supplementaryData, GraphMiningParametersSet miningParameters)
				throws GraphMiningException	{
		if (!(miningParameters instanceof GCE_ParametersSet)) {
				throw new PluginException("Mismatch of parameters type:
"+miningParameters.getAlgorithmName()+" parameters in "+

	Activator.getContext().getBundle().getSymbolicName());
		}
```

```
                InputStream gceOutput = GCEMiner.mine(graphOnDrive,
(GCE_ParametersSet) miningParameters);

                return new GCEResults(gceOutput, miningParameters);
        }

}


import java.io.IOException;
import java.io.InputStream;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.SAXException;

import ru.ispras.modis.NetBlox.exceptions.ScenarioException;
import ru.ispras.modis.NetBlox.parser.extensionInterfaces.IGraphMiningDescriptionParser;
import ru.ispras.modis.NetBlox.scenario.DescriptionGraphMiningAlgorithm;

public class GCEParserCallback implements IGraphMiningDescriptionParser {
        private final static GCEDescriptionParser parser = new GCEDescriptionParser();

        @Override
        public DescriptionGraphMiningAlgorithm parseMiningDescription(InputStream
tagContent) {
                SAXParserFactory saxfactory = SAXParserFactory.newInstance();
    saxfactory.setNamespaceAware(true);

    try {
                        saxfactory.newSAXParser().parse(tagContent, parser);
                } catch (SAXException | IOException | ParserConfigurationException e) {
                        throw new ScenarioException(e);
                }

                return parser.getParsedDescription();
        }
}



import java.io.InputStream;

import ru.ispras.modis.NetBlox.graphAlgorithms.graphMining.MinerResults;
import ru.ispras.modis.NetBlox.scenario.GraphMiningParametersSet;

public class GCEResults extends MinerResults {
        private InputStream inputStream;
```

```
	public GCEResults(InputStream gceOutput, GraphMiningParametersSet
miningParameters) {
		super(ResultsProvisionFormat.STREAM,
MinedResultType.NODES_GROUPS, miningParameters);

		inputStream = gceOutput;
	}

	@Override
	public InputStream getNodesGroupsStream()	{
		return inputStream;
	}
}


import java.util.ArrayList;
import java.util.List;

import ru.ispras.modis.NetBlox.scenario.GraphMiningParametersSet;
import ru.ispras.modis.NetBlox.utils.MiningJobBase;
import ru.ispras.modis.NetBlox.utils.Pair;

public class GCE_ParametersSet extends GraphMiningParametersSet {
	private int minimalCliqueSize = 4;

	private float minimalValueForOneSeedToOverlapWithAnotherSeed = (float) 0.6;
	private float alphaValue = (float) 1.0;
	private float phi = (float) 0.75;


	public GCE_ParametersSet(String algorithmNameInScenario, String
algorithmDescriptionID, int minimalCliqueSize) {
		super(MiningJobBase.JobBase.GRAPH, algorithmNameInScenario,
algorithmDescriptionID);
		this.minimalCliqueSize = minimalCliqueSize;
	}


	public int getMinimalCliqueSize()	{
		return minimalCliqueSize;
	}

	public float getMinimalValueForOneSeedToOverlapWithAnotherSeed()	{
		return minimalValueForOneSeedToOverlapWithAnotherSeed;
	}

	public float getAlphaValueForFitnessFunctionGreedilyExpandingCliques() {
		return alphaValue;
	}

	public float getProportionOfNodesWithinCoreCliqueForSufficientCoverage()	{
```

```
		return phi;
	}


	@Override
	public List<Pair<String, String>>
getSpecifiedParametersAsPairsOfUniqueKeysAndValues() {
		List<Pair<String, String>> result = new ArrayList<Pair<String, String>>(1);
		result.add(new Pair<String, String>("cl",
String.valueOf(minimalCliqueSize)));//XXX Change to reuse for generation?..
		return result;
	}
}


import java.util.ArrayList;
import java.util.Iterator;
import java.util.NoSuchElementException;

import org.osgi.framework.Bundle;

import ru.ispras.modis.NetBlox.scenario.DescriptionGraphMiningAlgorithm;
import ru.ispras.modis.NetBlox.scenario.ParametersSet;

public class DescriptionGCD_GCE extends DescriptionGraphMiningAlgorithm {
	private int minimalCliqueSize = 4;

	public void setMinimalCliqueSize(int size)	{
		minimalCliqueSize = size;
	}


	@Override
	protected Bundle getImplementingPluginBundle()		{
		return Activator.getContext().getBundle();
	}


	@Override
	public Iterator<ParametersSet> iterator() {
		if (doLaunchSeveralTimes())		{
			return new GCD_ParametersIterator();
		}

		ParametersSet parametersSet = new
GCE_ParametersSet(getNameInScenario(), getId(), minimalCliqueSize);
		ArrayList<ParametersSet> oneItemArray = new
ArrayList<ParametersSet>(1);
		oneItemArray.add(parametersSet);

		return oneItemArray.iterator();
```

```
	}


	/**
	 * Iterator over the combinations of GCE parameters.
	 *
	 * @author ilya
	 */
	private class GCD_ParametersIterator extends AlgorithmParametersIterator {
		@Override
		public ParametersSet next() {
			if (!resolveValues())		{
				throw new NoSuchElementException("There're no GCE
parameters available for next iteration as requested.");
			}

			GCE_ParametersSet parametersSet = new
GCE_ParametersSet(getNameInScenario(), getId(), minimalCliqueSize);

		parametersSet.setLaunchNumber(makeValueFromRangeInstance(launchNumbers,
launchNumber));
			return parametersSet;
		}
	}
}



import org.xml.sax.SAXException;

import ru.ispras.modis.NetBlox.parser.basicParsersAndUtils.Utils;
import ru.ispras.modis.NetBlox.parser.basicParsersAndUtils.XMLIntegerRangeStringProcessor;
import ru.ispras.modis.NetBlox.parser.xmlParser.CommonXMLParser;
import ru.ispras.modis.NetBlox.parser.xmlParser.XMLStringValueProcessor;
import ru.ispras.modis.NetBlox.scenario.RangeOfValues;

public class GCEDescriptionParser extends CommonXMLParser {
	class SupplementaryAlgosIdsProcessor extends XMLStringValueProcessor {
		@Override
		public void closeElement()  {
			super.closeElement();
			String stringOfIds = getText();
			String[] ids = stringOfIds.split(Utils.DELIMITER);
			for (String stringId : ids)	{
				Utils.checkWhetherIsWordInScenario(stringId,
TAG_SUPPLEMENTARY_ALGOS_IDS, "algorithm");


		minerDescription.addSupplementaryAlgorithmId(stringId);
			}
		}
```

```
	}

	class LaunchesProcessor extends XMLIntegerRangeStringProcessor		{
		@Override
		public void closeElement()  {
			super.closeElement();

			RangeOfValues<Integer> launchNumbers = getValues();
			if (launchNumbers != null  &&  !launchNumbers.isEmpty())		{
				minerDescription.setLaunchNumbers(launchNumbers);
			}
		}
	}


	private static final String TAG_SUPPLEMENTARY_ALGOS_IDS =
"supplementaryAlgosIds";
	private static final String TAG_LAUNCH_NUMBERS = "launchNumbers";
	private static final String TAG_MINIMAL_CLIQUE_SIZE = "minimalCliqueSize";

	private final XMLStringValueProcessor minimalCliqueSizeParser;

	private DescriptionGCD_GCE minerDescription;


	public GCEDescriptionParser()		{
		super();

		add(TAG_SUPPLEMENTARY_ALGOS_IDS, new
SupplementaryAlgosIdsProcessor());
		add(TAG_LAUNCH_NUMBERS, new LaunchesProcessor());
		add(TAG_MINIMAL_CLIQUE_SIZE, minimalCliqueSizeParser = new
XMLStringValueProcessor());
	}


	@Override
	public void startDocument() throws SAXException {
		super.startDocument();

		minerDescription = new DescriptionGCD_GCE();
	}


	@Override
	public void endDocument() throws SAXException		{
		super.endDocument();

		String text = minimalCliqueSizeParser.getText();
		if (text != null  &&  !text.isEmpty())  {
			minerDescription.setMinimalCliqueSize(Integer.parseInt(text));
```

```
                }
        }


        public DescriptionGCD_GCE getParsedDescription()    {
                return minerDescription;
        }
}


import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.util.LinkedList;
import java.util.List;

import org.apache.commons.lang.SystemUtils;

import ru.ispras.modis.NetBlox.exceptions.ExternalException;
import ru.ispras.modis.NetBlox.exceptions.GraphMiningException;
import ru.ispras.modis.NetBlox.graphAlgorithms.graphMining.GraphOnDrive;
import ru.ispras.modis.NetBlox.utils.ExternalApplicationProvider;

public class GCEMiner extends ExternalApplicationProvider {
        private static final String MINERS_ROOT;
        private static final String APPLICATION;
        static    {
                MINERS_ROOT =
getAppsForPluginRoot(Activator.getContext().getBundle(), "apps");

                if (SystemUtils.IS_OS_WINDOWS)  {
                        APPLICATION = MINERS_ROOT +
"GCECommunityFinder.exe";
                }
                else    {
                        APPLICATION = MINERS_ROOT + "GCECommunityFinder";
                }

        }


        public static InputStream mine(GraphOnDrive graphOnDrive, GCE_ParametersSet
parameters) throws GraphMiningException    {
                List<String> command = generateCommand(graphOnDrive, parameters);

                try {
                        PipedInputStream gceOutput = new PipedInputStream();
                        OutputStream processOutput = new
PipedOutputStream(gceOutput);
```

```
                runExternal(command, MINERS_ROOT, processOutput);

                return gceOutput;

            } catch (ExternalException e) {
                throw new GraphMiningException("Could not run GCE:
"+e.getMessage());
            } catch (IOException e) {
                throw new GraphMiningException("Could not process GCE output:
"+e.getMessage());
            }
        }


        private static List<String> generateCommand(GraphOnDrive graphOnDrive,
GCE_ParametersSet parameters)    {
            List<String> command = new LinkedList<String>();
            command.add(APPLICATION);
            command.add(graphOnDrive.getGraphFilePathString());
            command.add(String.valueOf(parameters.getMinimalCliqueSize()));

        command.add(String.valueOf(parameters.getMinimalValueForOneSeedToOverlapWit
hAnotherSeed()));

        command.add(String.valueOf(parameters.getAlphaValueForFitnessFunctionGreedilyE
xpandingCliques()));

        command.add(String.valueOf(parameters.getProportionOfNodesWithinCoreCliqueFor
SufficientCoverage()));

            return command;
        }
}
```

					ДП ІС-5103.1181-с.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера / Маса / Масштаб
Розробив		Борисенко А.Д.				
Перевірив		Попенко В.Д.				Аркуш 1 / Аркушів 1
Т. кон.					*Коплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51*
Н. кон.		Тєлишева Т.О.				
Затвердив		Попенко В.Д.				

ДП ІС-5103.1181-с.СБД

					ДП ІС-5103.1181-с.СБД	
Зм.	Арк.	№ документа	Підпис	Дата	Схема бази даних	Літера / Маса / Масштаб
Розробив		Борисенко А.Д.				
Перевірив		Попенко В.Д.				Аркуш 1 / Аркушів 1
Т. кон.					Коплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51
Н. кон.		Тєлишева Т.О.				
Затвердив		Попенко В.Д.				

					ДП ІС-5103.1181-с.ССП		
						Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовностей		
Розробив		Борисенко А.Д.				Аркуш 1	Аркушів 1
Перевірив		Попенко В.Д.					
Т. кон.					*Коплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51*	
Н. кон.		Тєлишева Т.О.					
Затвердив		Попенко В.Д.					

					ДП ІС-5103.1181-с.ССК	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Борисенко А.Д.				
Перевірив		Попенко В.Д.				Аркуш 1 / Аркушів 1
Т. кон.					Коплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51
Н. кон.		Тєлишева Т.О.				
Затвердив		Попенко В.Д.				

					ДП ІС-5103.1181-с.ССК		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Розробив		Борисенко А.Д.					
Перевірив		Попенко В.Д.				Аркуш 1	Аркушів 1
Т. кон.					*Коплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51*	
Н. кон.		Тєлишева Т.О.					
Затвердив		Попенко В.Д.					

Екранна форма головної сторінки

Екранна форма, де демонструється робота алгоритмів на реальній мережі

Екранна форма, де демонструється порівняльний аналіз двох алгоритмів на згенерованій мережі

					ДП ІС-5103.1181-с.КЕ		
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера	Маса / Масштаб
Розробив		Борисенко А.Д.					
Перевірив		Попенко В.Д.				Аркуш 1	Аркушів 1
Т. кон.					Коплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51	
Н. кон.		Тєлишева Т.О.					
Затвердив		Попенко В.Д.					

Рішення з математичного забезпечення

Задача виявлення спільнот.

Нехай є мережа G, яка має n вершин та m ребер. Та спільнота C, яка має n_C вершин та m_C ребер.

Розрахуємо щільність графу:

$$\rho = \frac{2m}{\left(n(n-1)\right)}. \quad (3.1)$$

Щільність внутрішніх зв'язків спільнот:

$$\delta_{int}(C) > \rho. \quad (3.2)$$

Щільність зовнішніх зв'язків:

$$\delta_{ext}(C) = \frac{m_{ext}}{\left(n_C(n-n_C)\right)}, \quad (3.3)$$

$$\delta_{ext}(C) < \rho, \quad (3.4)$$

де m_{ext} – кількість зовнішніх ребер.

Завдання виявлення спільнот полягає в максимізації суми різниць щільності $(\delta_{int} - \delta_{ext})$ по всіх спільнотах мережі.

Демонстраційний плакат до дипломного проекту „Комплекс задач для порівняльного аналізу алгоритмів виявлення прихованих підгруп у соціальній мережі ”	
Виконав студент гр. ІС-51	Борисенко А.Д.
Керівник ДП	Попенко В.Д.