ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних систем та автоматики
Кафедра комп’ютерних систем управління
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма Інтелектуальні комп’ютерні системи

ЗАТВЕРДЖУЮ

Завідувач кафедри КСУ
Дубовой В.М.

«___» _____ 2019 року

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

Розробка програмного забезпечення системи координнаційного управління. Частина 2. Однорівнева система управління

08-01.МКР.011.00.000

Студент групи 2АКІТ-18м
Шевчук О. І.
Керівник к.т.н., доцент
Юхимчук М. С.
Рецензент к.т.н., доцент
Маслій Р. В.

Вінниця ВНТУ – 2019

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних систем та автоматики
Кафедра комп'ютерних систем управління
Освітньо-кваліфікаційний рівень магістр
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані
Освітньо-професійна програма Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
Дубовой В.М.

«_2__» ____09_____ 2019року

Протокол № 1 засідання кафедри КСУ від 2.09.2019р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____Шевчук Олег Ігорович_____
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Розробка програмного забезпечення системи координнаційного управління. Частина 2. Однорівнева система управління»

керівник магістерської кваліфікаційної роботи Юхимчук Марія Сергіївна,

д. т. н., доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "02" 10 2019 року № 254

2. Строк подання студентом магістерської кваліфікаційної роботи 10.12. 2019 року

3. Вихідні дані до магістерської кваліфікаційної роботи найкращий Y_{out} та підібрані β після здійснення оптимізації; підтримка ОС – Windows, Android; Linux, IOS, максимальний час завантаження – 5 с; авторизація користувачів – ні; мова графічного інтерфейсу – українська,.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) вступ, аналіз варіантів побудови архітектур систем координаційного управління, розробка критеріїв для порівняння та порівняння за цими критеріями архітектур систем, розробка алгоритму знаходження для кожної із точок керування однорівневою системою управління таких сусідніх точок, на які поширюється вплив згідно

із заданим законом поширення та розробка хвильового алгоритму, практична реалізація та тестування системи моделювання координаційного управління та економічна частина.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) блок-схема алгоритму знаходження для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення., UML діаграма варіантів використання, UML-діаграма послідовності, результати оптимізації.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ змістової частини роботи	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	канд. економ. наук, професор кафедри ЕПВМ Козловський В.О		

Календарний план

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1	Вступ. Постановка задач дослідження		
2	Змістова частина роботи (спеціальні науково-дослідні, дослідно-конструкторські, дослідно-технологічні та технічно-розрахункові розділи).		
3	Практична реалізація та аналіз отриманих результатів		
4	Економічний розділ		
5	Апробація результатів дослідження		
6	Публікації		
7	Оформлення пояснювальної записки, графічного матеріалу і презентації		
8	Захист МКР		

Дата видачі завдання "_02__" __09___2019 року

Студент _____ _Шевчук О. І.

Керівник магістерської кваліфікаційної роботи _____ _Юхимчук М. С.

АНОТАЦІЯ

Дана магістерська робота присвячена розробці програмного забезпечення однорівневого координаційного управління. Проведено аналіз предметної області та варіантний аналіз методів створення архітектури систем. Також було розроблено та реалізовано алгоритм методу знаходження для кожної із точок керування однорівневою системою управління таких сусідніх точок, на які поширюється вплив згідно із заданим законом поширення, а також хвильовий алгоритм проходження вершин графа. Розроблено UML діаграми та супровідну документацію. Розроблене програмне забезпечення пройшло тестування, яке підтвердило правильність його роботи.

ABSTRACT

This master's work is devoted to the development of one-tier coordination control software. The analysis of the subject domain and variant analysis methods of creation architecture of system are carried out. An algorithm of the method of finding for each of the control points of the one-tier control system of such adjacent points, which is affected by the law of distribution and the wave algorithm of passing the vertices of the graph, was also developed and implemented. UML diagrams and supporting documentation have been developed. The developed software was tested, which confirmed the integrity of its work.

ЗМІСТ

ВСТУП

Актуальність проблеми. Бурхливий розвиток комп'ютерних технологій, системного аналізу та теорії автоматичного управління дозволяють в даний момент вирішувати цілий ряд складних питань, що на пряму пов'язані із автоматичним та автоматизованим управлінням великими технологічними об'єктами та цілими виробничими комплексами. Прикладами вирішення подібних задач виступають як автоматизовані системи управління великою кількістю енергетичних, металургійних чи хімічних виробництв, так і архітектурні рішення побудови цих систем. При розробці таких систем одними із першочергових задач постають задачі вибору ефективних організаційних структур, розподілення як і в середині них, так і між ними прав і обов'язків координації і управління їх спільною діяльністю. Безумовно, дослідження на дану тематику є актуальними та необхідними, адже вони сприяють підвищенню якості управління великими динамічними системами, що складаються із багатьох компонентів. У даній комплексній магістерській роботі досліджуються варіанти побудови існуючих систем координаційного управління та розроблено програмне забезпечення моделювання різних підходів до координаційного управління, а саме однорівневого та ієрархічного.

Тема комплексної роботи – розробка програмного забезпечення системи координаційного управління.

Тема даної роботи (друга частина комплексної роботи): Частина 2. Однорівнева система управління.

Метою магістерської кваліфікаційної роботи є підвищення якості рішень при однорівневому координаційному управлінні.

Об'єкт дослідження – процес моделювання та функціонування однорівневої координаційної системи управління.

Предмет дослідження – методи оптимального управління координаційною системою із урахуванням її архітектури.

Іноваційність одержаних результатів.

1. Розроблено алгоритм знаходження для кожної із точок керування однорівневою системою управління таких сусідніх точок, на які поширюється вплив згідно із заданим законом поширення.

2. Удосконалено хвильовий алгоритм для проходження через усі вершини графа, що базується на використанні пошуку в ширину.

Практичне значення одержаних результатів полягає в першу чергу в отриманні програмної системи, що дозволяє промоделювати управління у однорівневій системі координаційного управління, а також дає змогу побудувати та візуалізувати граф сусідства точок керування, що відображає вплив та зв’язок між ними.

Всі розроблені програмні модулі успішно пройшли тестування, яке підтвердило коректність і ефективність розробленої системи.

Апробація результатів. Основні положення та результати виконаних в магістерській роботі досліджень доповідались та обговорювались на конференціях:

– XLVIII Науково-технічна конференція підрозділів Вінницького національного технічного університету (2019) [57].

По розробленому програмному забезпеченню було подано заявку на авторський твір.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Системи управління

Контроль – це засіб впливу на об’єкт, щоб він поводився бажаним чином. Об'єктом може бути економічна система, технологічний процес (наприклад, хімічний завод), система водопостачання або екологічна система [1]. Таким чином, проблеми управління були розроблені в багатьох сферах, виникли різні термінології, і були розроблені різні підходи. Отже, варто спочатку визначити деякі основні поняття та формулювання, що стосуються систем управління.

На рисунку 1.1 схематично представлена типова система управління, що показує керовану систему (її також називають процес, установка, об'єкт, середовище) та блок управління (контролер, блок прийняття рішень). Керована система має деякі керовані входи (позначені вектором m), що можуть спричиняти зміну виходів y (також звані результатами). Керована система також підпорядковується іншій групі входів, які перебувають поза нашим впливом і називаються збурення z. Збурення слід по суті вважати випадковими змінними.

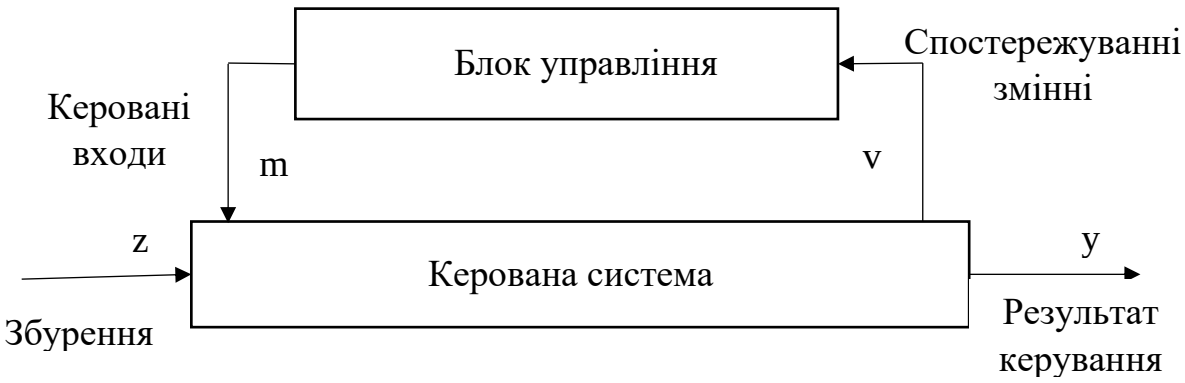


Рисунок 1.1 – Типова система управління

Завданням блоку управління (який у складному випадку може складатися із декількох частин) – визначити значення m, для досягання певної мети, тобто, що відповідає певній специфікації щодо поведінки керованої системи [2]. Ми називаємо значення, визначені блоком управління, як рішення управління або просто керування. Блок управління засновує свої поточні рішення управління на спостережуваних змінних або спостереженнях (вимірювання, позначені v), які пов'язані з керованою системою або збуренням або ними обома [3]. Формуючи керуючі рішення, блок управління разом із спостереженнями використовує своє розуміння керованої системи. Під розумінням керованої системи мається на увазі модель системи.

1.2 Приклади однорівневих систем

Технологія p2p не є новою та існує із часів появи децентралізованих мереж – Usenet та FidoNet.

У 1979 році для обміну інформацією в групах новин було створено Usenet. Обмін інформацією відбувався телефонними лініями задля економії переважно вночі, тому що це було дешевше [4].

FidoNet, — це децентралізована, розподілена мережа для обміну повідомленнями, створена у 1984 році Томом Дженнінгсом для обміну повідомленнями між користувачами різних електронних дошок оголошень. FidoNet існує до сьогоднішнього дня [5].

Виділяють три покоління пірингових мереж, наведемо їх короткий опис.

1.2.1 Перше покоління

Перше покоління p2p мереж характеризується наявністю виділених центральних серверів, що могли бути базами даних та займатися координацією пошуку. Проте архітектура таких мереж дозволяє зв'язок та передачу інформації безпосередньо між будь-якими її учасникам .[6]

Широку популярність peer-to-peer отримав після створення мережі Napster[7]. Napster - це система обміну файлами, що дозволяє користувачам шукати та завантажувати музичні файли, що зберігаються на жорстких дисках користувачів Napster [8]. Схему роботи мережі Napster зображено на рисунку 1.2.

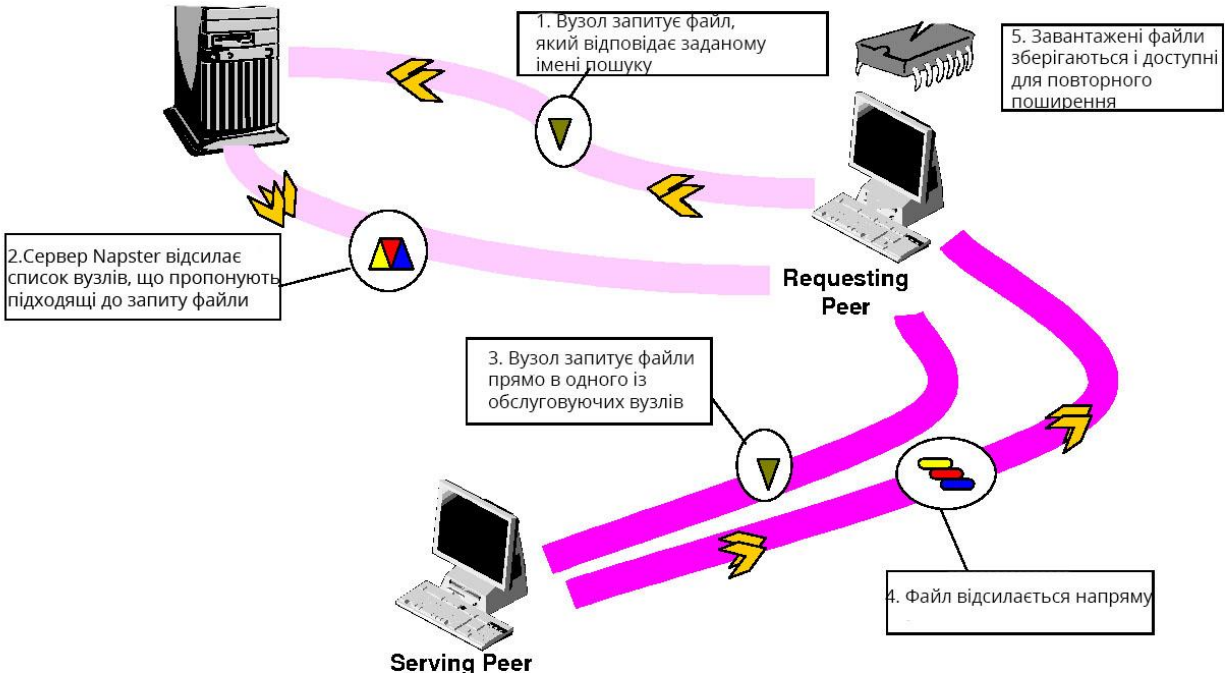


Рисунок 1.2 – Схема роботи мережі Napster

Коли програма запускається вперше, метадані для всіх музичних файлів користувача переноситься у глобальний каталог. Коли інші користувачі шукають пісню за допомогою ключових слів у цих метаданих, каталог повертає список клієнтів, які діляться піснями, що відповідають запиту. Кінцеві машини (в цьому сенсі вузли) співпрацюють, щоб передати пісню безпосередньо між собою. Кожен бере на себе роль клієнта та сервера, чергуючись, вони завантажують пісню іншому користувачеві чи отримують нову для себе. Думки відрізняються щодо того, чи можна систему Napster визначити справді одноранговою, оскільки її каталог зберігається на центральних серверах, і клієнти ніколи не беруть участі в обробці пошукових запитів [9]. Однак, поширивши пропускну здатність і вимоги до зберігання

самих музичних файлів, системі вдалося покращити свої початкові недоліки. У технічному плані програма зазнала простого інтерфейсу, і низька надійність та пропускна спроможність з'єднань інших клієнтів часто гальмували спроби користувачів отримувати пісні. Однак, оскільки вона різко спростила завдання отримання музики в Інтернеті (хоча й переважно нелегально), вона стала надзвичайно популярною – на своєму піку Napster може похвалитися близько 1,6 мільйона одночасних користувачів [10].

1.2.2 Друге покоління: повна децентралізація

Першим представником був Gnutella [11], розподілений протокол пошуку, прийнятий декількома програмами обміну файлами, які не використовували централізований каталог і замість цього транслювали пошукові запити між сусідніми вузлами. Проте тестування та досвід користувачів свідчать, що як запити, так і обсяг повідомлень викликали надмірне навантаження на мережі, обмежуючи розмір мереж Gnutella, та зменшуючи шанс задовольнити заданий запит та значне послаблення пропускної здатності клієнта, що залишилася для фактичної передачі файлів [12].

Пізніше, системи для розміщення контенту, включаючи Freenet, додали механізми для маршрутизації запитів до відповідного вузла, в якому, швидше за все, знаходився необхідний вміст [13]. Типову схему однорангової мережі другого покоління зображено на рисунку 1.3.

Інші системи для спільного використання файлів, такі як Morpheus [14] та Reflector Clip2 [15], додали структуру мережам спільного доступу до файлів однорангових мереж шляхом динамічного вибору вузлів, які ставали супер-вузлами, кешували та обслуговували загальні запити чи надавали контент. А самі звичайні вузли здійснювали запити та надавали вміст через виділеного супер-вузла. Ці схеми використовують перевагу спостережуваного Zipf-

розподілу запитів на вміст та зменшують труднощі з передачею запитів через хости з високою затримкою, низькою комутацією пропускної здатності [16].

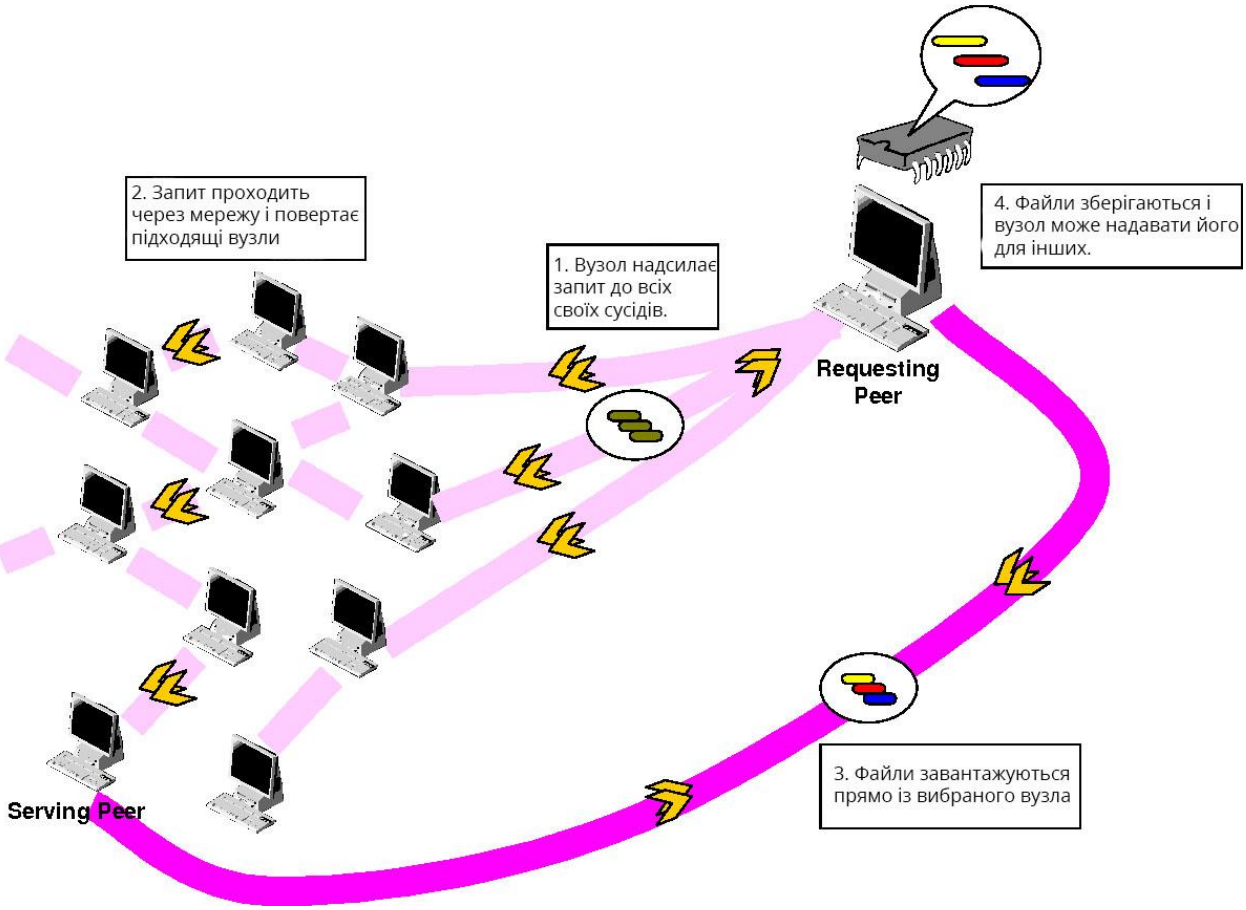


Рисунок 1.3 – Типова схема однорангової мережі другого покоління

1.2.3 Третє покоління однорангових мереж

Для того, щоб кожен вузол зробив корисний внесок у глобальну мережу, потрібен надійний спосіб розподілу робочого навантаження та адресації відповідального вузла. В основі систем однорангових мереж третього покоління лежить децентралізована структура та розподілені хеш-таблиці (DHT), де вузлам присвоюється унікальний псевдовипадковий ідентифікатор, який визначає їх положення в ключовому просторі [17]. Типову схему однорангової мережі третього покоління зображено на рисунку 1.4

Для ефективного отримання файлів в мережі в різних вузлах для індифікування файлів індексуються деякі значення хеш-функції, чим самим вирішується проблема масштабування [18].

Розвиток цих мереж призвів до збільшення кількості та різноманітності розроблених однорангових програм. З'явилися системи для розповсюдження подій та масової багатоадресної розсилки, архівування файлів, файлові системи та програми для заміни DNS [19].

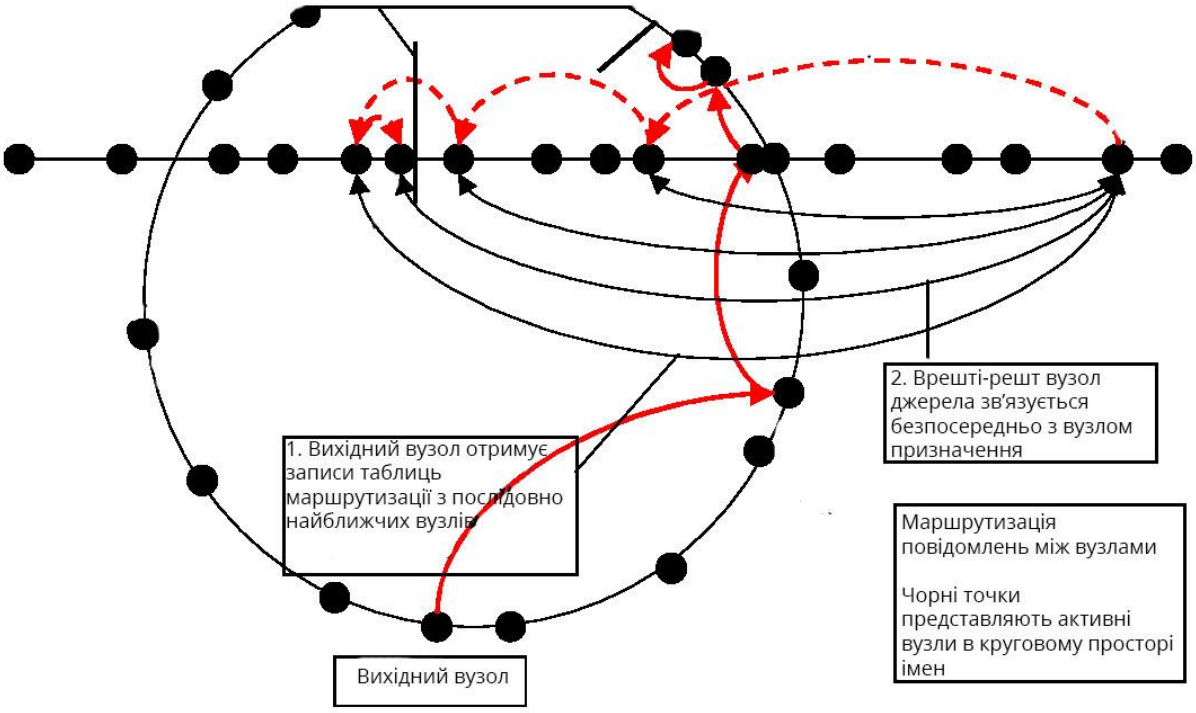


Рисунок 1.4 – Типова схема однорангової мережі третього покоління

1.3 Однорівневі системи управління в розподілених обчисленнях

Розподілена обчислювальна система визначається як сукупність незалежних комп'ютерів, які постають перед їх користувачами як єдина обчислювальна система [20]. Розподілені програмні системи все частіше використовуються у розробці сучасних програмних систем для вирішення питань географічно відокремлених робочих груп та збільшення складності прикладних програм. Часто це являє собою взаємозв'язок автономного програмного забезпечення, що знаходиться на окремих машинах, через комунікаційні мережі, що дозволяє їхнім користувачам співпрацювати та координуватись для успішного виконання своїх цілей.

Природним розширенням філософії надійності Інтернету через децентралізацію є розробка систем однорангових мереж, що надсилають обчислювальні завдання на мільйони серверів, кожен з яких, можливо, також є настільним комп'ютером.

Поширена потреба у розподілених системах обумовлена потребою в обміні ресурсами та відмовостійкості. Обмін ресурсами означає, що розподілена система дозволяє належним чином розподіляти між своїми користувачами ресурси – обладнання, програмне забезпечення та дані [21]. Толерантність до відмов означає здатність машин підключених мережами протистояти несправностям обладнання або збоям програмного забезпечення шляхом встановлення програмного забезпечення на декількох машинах[22].

Щоб розподілена система підтримувала активний обмін ресурсами та стійкість до відмов у своїй безлічі вузлів, вона повинна володіти певними ключовими властивостями. До них належать відкритість та прозорість. Відкритість в розподіленій системі досягається шляхом визначення її ключових елементів інтерфейсу та надання доступу до них для інших розробників програмного забезпечення, щоб система могла бути розширена для використання [23]. Розподілені системи, як правило, забезпечують три форми прозорості. До них належать:

1. Прозорість місцеположення, яка дозволяє отримати доступ до місцевої та віддаленої інформації уніфікованим способом;

2. Прозорість відмов, що дозволяє автоматично замаскувати відмови;

3. Прозорість реплікації, яка дозволяє непомітно дублювати програмне забезпечення або дані на декількох машинах.

Щоб мати змогу продемонструвати раніше згадані властивості, у свою чергу така розподілена система повинна забезпечувати підтримку паралельності та будуватися на масштабованій архітектурній основі. Паралельність означає одночасну обробку запитів на декількох взаємопов'язаних машинах чи мережах [24]. Масштабованість слід враховувати при розробці архітектури мереж взаємозв'язку, що дозволить

безперешкодно розширювати кількість машин та користувачів для підтримки потреб у підвищені необхідної обчислювальної потужності.

У розподілених системах, що базуються на архітектурі p2p виділяють три конкретні категорії додатків p2p [25]:

1. Паралельні програми: У таких додатках великий обчислення розбивається на кілька невеликих незалежних об'єктів, які можна виконати самостійно на великій кількості вузлів, що дає можливість зробити обчислення однієї операції, але з різними наборами даних або параметрами. Такий вид обчислень називають параметричним обчисленням дослідження.

2. Управління контентом і файлами: контент інкапсулює декілька видів діяльності і посилається на все, що може бути оцифровано; наприклад, повідомлення, файли, двійкове програмне забезпечення. Він по суті складається із зберігання, обміну та пошуку різного роду інформації в мережі. Основна увага в програмі - обмін контентом. Такі проекти мають на меті створити систему файлів для розповсюдження в межах певної спільноти, інформація про вузли такої системи знаходиться в розподілених базах даних та розподілених хеш-таблицях [26].

3. Співпраця: Спільні програми P2P дозволяють користувачам співпрацювати в режимі реального часу, не покладаючись на центральний сервер для збору та ретрансляції інформації. Такі програми характеризуються постійними взаємодіями та обмінами між вузлами. Типові програми включають: обмін миттєвими повідомленнями (AOL, YM!). Деякі ігри (DOOM) також прийняли рамки P2P [27].

1.4 Обґрунтування критеріїв для порівняння існуючих архітектур створення систем

Для порівняння сучасних варіантів архітектур при розробці систем слід обрати такі критерії порівняння, що будуть найкраще відображати переваги та

недоліки кожної. Критерії порівняння мають продемонструвати які підходи краще використовувати в яких ситуаціях і які в них є можливості.

Виділимо основні критерії для порівняння архітектур систем:

1. Спільне використання ресурсів. Архітектура системи надає можливість для спільного користування апаратних та програмних ресурсів, наприклад жорстких дисків, принтерів та файлів, що зв’язані мережею.

2. Відкритість, – це властивість, яка дає змогу збільшувати систему шляхом додавання нових ресурсів.

3. Паралельність. Одночасне виконання кількох процесів на різних машинах в мережі із можливістю взаємодії процесів один із одним під час виконання.

4. Масштабування. З часом для того, щоб система була конкурентною та могла виконувати більш складні завдання її можна підсилити за допомогою додавання додаткових обчислювальних ресурсів. Проте в дійсності розширення може обмежуватися мережею, що об’єднує окремі компоненти системи. При підключенні багатьох нових членів мережі, її пропускна можливість може сильно зменшитися.

5. Відмовостійкість. Дана властивість при виникненні різного роду помилок дозволяє системі залишатися стійкою, що забезпечується можливістю дублювання інформації на декількох машинах.

6. Прозорість. Користувач має можливість повного доступу до ресурсів, проте йому не відомо про розподілення ресурсів у системі.

1.5 Варіантний аналіз методів створення архітектури систем

На даний час існує багато варіантів архітектур систем, проте в даній роботі розглядається лише декілька самих популярних із них, а саме: дворівнева клієнт-серверна архітектура, трирівнева клієнт-серверна архітектура, grid системи та однорангова архітектура.

1.5.1 Дворівнева клієнт-серверна архітектура

Дворівневі архітектура була розроблена в 1980-х роках з дизайну архітектури програмного забезпечення файлового сервера. Дворівнева архітектура призначена для покращення зручності використання, підтримує зручний інтерфейс, заснований на формах. Дворівнева архітектура покращує масштабованість за рахунок розміщення до 100 користувачів (архітектури файлових серверів вміщують лише десяток користувачів) та покращує гнучкість, дозволяючи ділитися даними, як правило, в однорідному середовищі. Дворівнева архітектура частіше застосовується в не складних системах обробки інформації, наприклад в системах підтримки прийняття рішень, де навантаження на транзакції невелике [28].

Дворівнева архітектура складається з трьох компонентів, розподілених у два шари: клієнт (запитувач послуг) та сервер (постачальник послуг). Наведемо перелік цих компонентів:

1. Інтерфейс користувача (наприклад, сеанс, введення тексту, діалог та послуги управління дисплеєм)

2. Управління процесами (такі як розробка процесів, впровадження процесів та моніторинг процесів)

3. Управління базами даних

Дворівнева система складається з клієнта та сервера. У дворівневій системі база даних зберігається на сервері, а інтерфейс, який використовується для доступу до бази даних, встановлюється на клієнті. Користувальницький системний інтерфейс зазвичай розташований у середовищі робочого столу користувача, а служби управління базами даних зазвичай на сервері, який є більш потужною машиною, яка обслуговує багатьох клієнтів. Управління процесами розбито між середовищем інтерфейсу користувача і середовищем сервера баз даних. Сервер баз даних забезпечує можливість створення збережених процедур та тригерів [29]. Схематичне представлення раніше описаних функцій дворівневої системи зображено на рисунку 1.5.

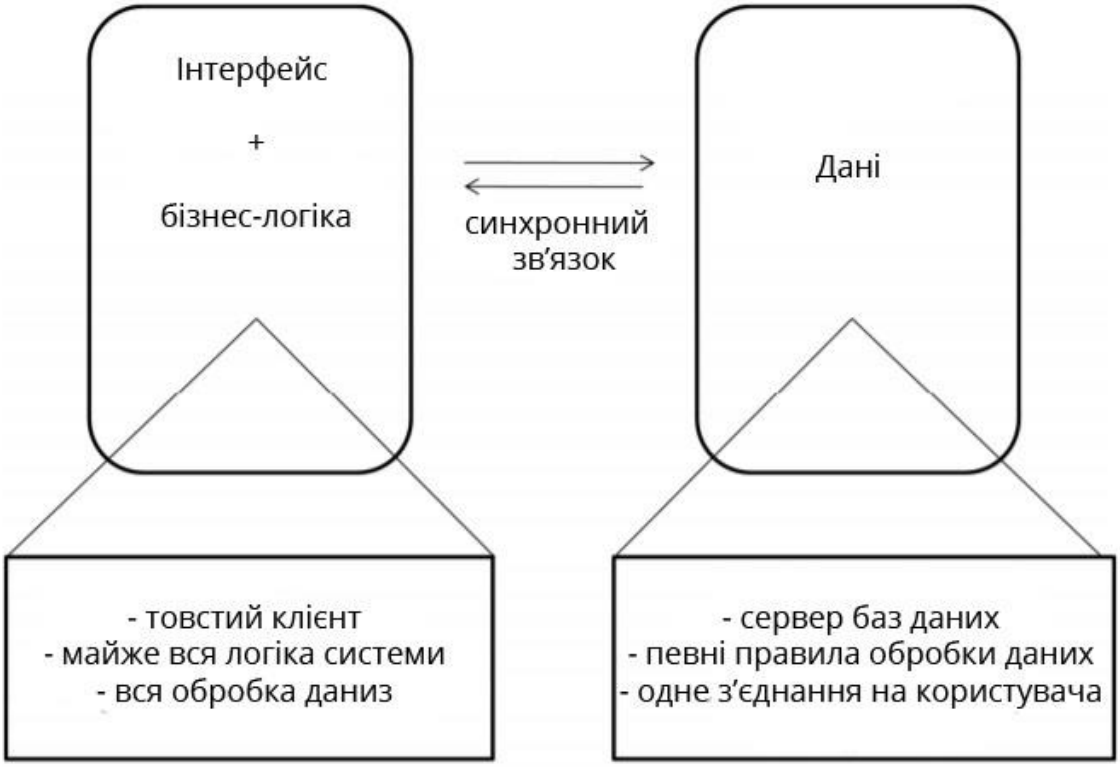


Рисунок 1.5 – Схематичне представлення дворівневої клієнт-серверної системи

До переваг систем даного типу можна віднести простоту реалізації і оскільки обробка поділялася між клієнтом і сервером, більше користувачів могли взаємодіяти з такою системою.

Недоліки:

– коли кількість користувачів перевищує 100, продуктивність починає погіршуватися. Це обмеження є результатом того, що сервер підтримує з'єднання за допомогою "підтримання живих" повідомлень з кожним клієнтом, навіть коли робота не виконується.

– другим обмеженням дворівневої архітектури є те, що реалізація служб управління обробкою з використанням власних процедур бази даних серверів обмежує гнучкість та вибір СУБД для додатків.

– поточні реалізації дворівневої архітектури забезпечують обмежену гнучкість у переміщенні (перерозподілі) функціональності програми з одного сервера на інший без регенерації процедурного коду вручну [30].

1.5.2 Трирівнева клієнт-серверна архітектура

Трирівнева архітектура систем виникла в 90-х роках для подолання згаданих в розділі 2.2.1 обмежень дворівневої архітектури. Третій рівень (сервер середнього рівня) знаходиться між користувальницьким інтерфейсом (клієнтом) та компонентами управління даними (сервер). Цей середній рівень забезпечує процес управління, де виконуються бізнес-логіка та діють правила, які можуть вмістити сотні користувачів (порівняно лише зі 100 користувачами з дворівневою архітектурою), забезпечуючи такі функції, як черга, виконання додатків та встановлення баз даних. Трирівнева архітектура використовується тоді, коли потрібна ефективна розподілена клієнт-серверна система, яка забезпечує (порівняно з двома рівнями) підвищену продуктивність, гнучкість, ремонтопридатність, повторне використання та масштабованість, приховуючи при цьому складність розподіленої обробки від користувача [31].

Трирівнева розподілена архітектура клієнт-сервер (як показано на рисунку 1.6) включає верхній рівень інтерфейсу користувальницької системи, де розміщуються сервіси користувачів (такі як сеанс, введення тексту, діалог та управління дисплеєм).

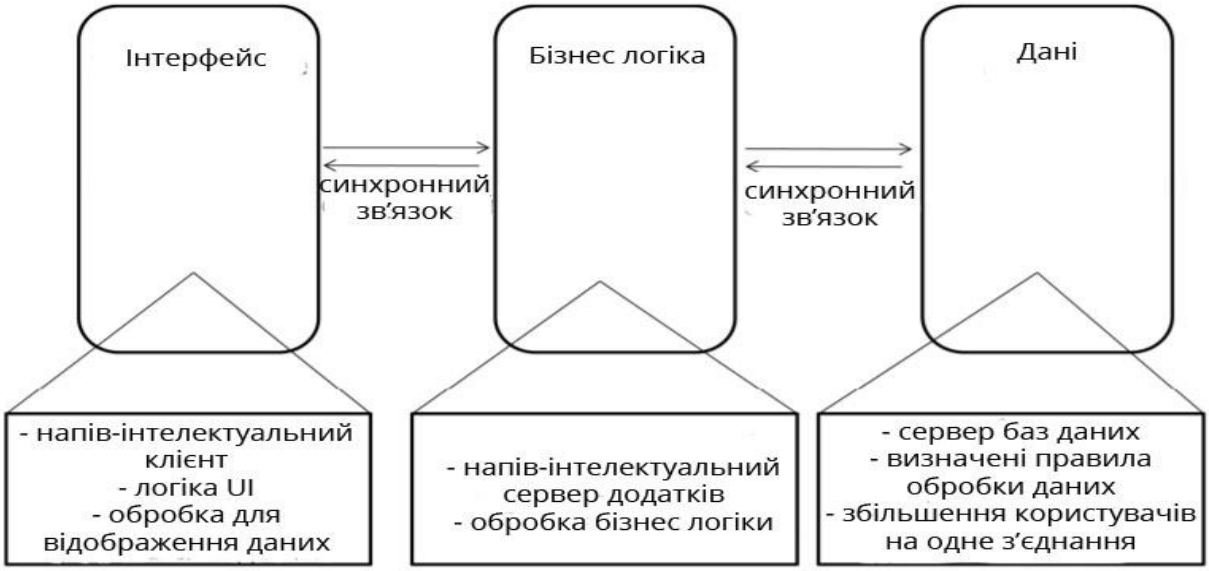


Рисунок 1.6 – Схематичне представлення трирівневої клієнт-серверної системи

Третій рівень забезпечує функціональність управління базами даних і призначений для служб даних та файлів, які можна оптимізувати без використання будь-яких власних систем управління базами даних. Компонент управління даними забезпечує узгодженість даних у всьому розподіленому середовищі через використання таких функцій, як блокування даних, узгодженість та реплікація. Слід зазначити, що зв’язок між рівнями може бути динамічно змінений залежно від запиту користувача на дані та послуги [32].

Середній рівень забезпечує послуги з управління процесами (такі як розробка процесів, запровадження процесів, моніторинг процесів і ресурсне забезпечення процесів), які поділяються кількома програмами.

Сервер середнього рівня (також його називають сервером додатків) покращує продуктивність, гнучкість, ремонтопридатність, повторне використання та масштабованість за допомогою централізації логіки процесу.

Централізована логіка процесу полегшує управління та реакцію на зміни в системі, локалізуючи функціональність системи, так що зміни потрібно записати лише один раз та розмістити на сервері середнього рівня, для доступу усіх систем.

Переваги:

Трирівнева архітектура клієнт-серверної системи покращує продуктивність для груп із великою кількістю користувачів та підвищує гнучкість у порівнянні з дворівневою системою.

Застосування даного підходу дає змогу розподілити рівень бізнес логіки серед декількох компонентів системи(частина на клієнті, частина на сервері) та зменшити кількість проблем при розгортанні систем шляхом централізації більшої кількості логіки на серверах.

Серверні компоненти, що перейшли на виділені сервера системи, забезпечили можливість управління пулами з’єднання із базами даних, цим самим полегшили задачу серверу баз даних шляхом значного зменшення одночасної кількості з’єднань, так як одне з’єднання може забезпечити роботу декількох клієнтів [33].

Масштабованість - кожен рівень може масштабуватися горизонтально. Наприклад, можна завантажити баланс рівня презентації між трьома серверами, щоб задовольнити більше веб-запитів, не додаючи сервери в рівні додатків і даних.

Продуктивність – оскільки рівень презентації може кешувати запити, використання мережі зводиться до мінімуму, а завантаження на рівні додатків і даних зменшується.

Доступність – якщо сервер рівня додатків не працює і кешування достатньо, рівень рівня презентації може обробляти веб-запити за допомогою кешу.

Недоліком даної системи є те, що дані системи важче розробляти ніж дворівневі.

1.5.3 Grid системи

Основна ідея grid - це віддалене централізоване представлення ресурсів, необхідних для вирішення всіляких завдань. Можна призначити будь-яке завдання комп'ютеру або мобільному пристрою, але ресурси для обчислення повинені бути автоматично надані на віддалені потужні сервери, незалежно від типу. завдання [34].

З більш практичної сторони, головним завданням, що стоїть за концепцією grid, є узгодження розподілених ресурсів та вирішення завдань у контексті динамічних та багатопрофільних віртуальних організацій [35]. Обмін ресурсами, що цікавить розробників Grid, не є файловим обміном, а безпосередньо доступ до комп'ютерів, пакетного програмного забезпечення, даних та інших ресурсів, які потрібні для спільного вирішення проблем та стратегічного управління ресурсами, що виникають в промисловості, науці та техніці [36].

Віртуальною організацією називають низку окремих осіб чи закладів, об'єднаних єдиним правилом колективного доступу до розрізнених

обчислювальних ресурсів [37]. Різницю між grid та p2p системами відображено у таблиці 1.1

Таблиця 1.1 – Порівняння Grid та p2p систем

Критерії	Grid	p2p
Цільова аудиторія	Орієнтований на потреби підприємств та наукових закладів.	Використовується в файлообмінниках і високопаралельних обчисленнях
Додатки	Орієнтовані на наукові додатки, потребують великої інтенсивності обміну даними	Орієнтовані на додатки обміну файлами та взаємодії користувачів
Масштаб і відмовостійкість	Вміщують в себе середню кількість користувачів, слабка відмовостійкість	Вміщують в себе мільйони користувачів за рахунок яких мають високу відмовостійкість
Служби і інфраструктура	Стандартизований стандартом OGSA	Орієнтуються на інтеграцію простих протоколів окремих ресурсів

1.5.4 Відмінності однорангових систем від клієнт-серверної моделі

Уже більше 25 років однорангові системи протиставляються традиційним клієнт-серверним системам, кожна з яких має як і слабкі, так і сильні сторони.

Клієнт-серверні системи несиметричні; ми зазвичай припускаємо, що сервер – це набагато більш потужна, краще підключена машина [38]. Сервер відрізняється тим, що працює протягом деякого більш тривалого періоду часу та керує ресурсами, зберіганням та обчислювальною роботою для деякої

кількості клієнтів. Як такий, сервер постає як єдине вузьке місце для продуктивності та надійності. При розробці серверів можуть використовувати низку методів для пом'якшення цих проблем, таких як реплікація, балансування завантаження та маршрутизація запитів, так що один сервер в деяких випадках складається з безлічі різних машин [39]. У якийсь момент еволюції цього мислення природним кроком є включення ресурсів клієнтів у систему – взаємні вигоди у великій системі зрозумілі. Розрив у продуктивності між настільними та серверними машинами зменшується, а поширення широкосмугового зв’язку значно покращує підключення кінцевих клієнтів. Таким чином, однорангові системи виходять із систем клієнт-сервер, видаляючи асиметрію в ролях: клієнт також є сервером і дозволяє отримувати доступ до своїх ресурсів іншими системами [40]. Клієнти, тепер справді рівноправні, вносять свої власні ресурси в обмін на власне використання послуги. Робота (будь то передача повідомлень, обчислення, зберігання чи пошук) розподіляється в деяких (зазвичай розподілених) засобах між усіма вузлами, так що кожен вузол споживає власні ресурси від імені інших (виступаючи як сервер), але так, щоб він може попросити інші вузли зробити те саме для цього (виступаючи як клієнт).

Можна подивитися на відмінності та схожість між класичними клієнт-серверними та сучасними peer-to-peer системами із сторони пильності. Незважаючи на успіх на серверах без стану, багато веб-серверів пропонують більш широкі функціональні можливості, використовуючи файли cookie, керовані скриптами сховища та веб-сервіси для збереження стану за різними транзакціями з клієнтом. У одноранговій системі, оскільки вузлу рідко відомо, який вузол здатний виконати його запит, кожен відслідковує стан сусідніх вузлів для забезпечення передачі запитів, повідомлень або статусу по мережі [41].

Ці системи можна розділити ще по одному важливому критерію, – це використання посередників. У веб (та файлових системах клієнт-сервер, таких як NFS та AFS) ми використовуємо кеш для поліпшення середньої затримки

та зменшення навантаження на мережу, але зазвичай вони розташовані статично. В однорангових системах динамічний розподіл роботи між зкооперованими вузлами дозволяє чудовий локально-орієнтований баланс навантаження. Беручи приклад кешування, системи розподілу контенту, такі як PAST та Pasta , мають на меті поширювати дані пропорційно до попиту на них, на вузлах, близьких у мережі до цього попиту [42].

2 РОЗРОБКА АЛГОРИТМІВ ТА UML ДІАГРАМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Розробка методу знаходження для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення.

Для того, щоб параметри об’єкта керування знаходилися в заданих рамках використовуються регулятори, які здійснюють вплив на ці об’єкти, зазвичай цей вплив націлений на конкретні точки(елементи). Система координації задає для цих регуляторів оптимальні параметри.

Модель системи складається з компонентів (рис. 2.1) [56]:

1. Модель об’єкта складається із двох залежностей: залежність між власними параметрами об’єкта – $\overline{O}_{0i,j}(P_{i,j}) = 0$, та залежність параметрів об’єкта керування від впливу параметрів сусідніх об’єктів керування – $P_{i,j} = \overline{\overline{O}}_{i,j}(V_{i,j}, P_{i-1,j}, P_{i+1,j}, P_{i,j-1}, P_{i,j+1})$, де індекси – дискретизовані і пронумеровані значення координат;

2. Модель керування $V_{i,j} = A_{i,j}(P_{i,j}, U_{i,j})$, де $U_{i,j}$ - параметри координації;

3. Модель координації $U_{i,j} = C_{i,j}(P_{i-a,j}, P_{i+b,j}, P_{i,j-c}, P_{i,j+d})$, де a,b,c,d - дискретизовані відстані між елементами (i, j).

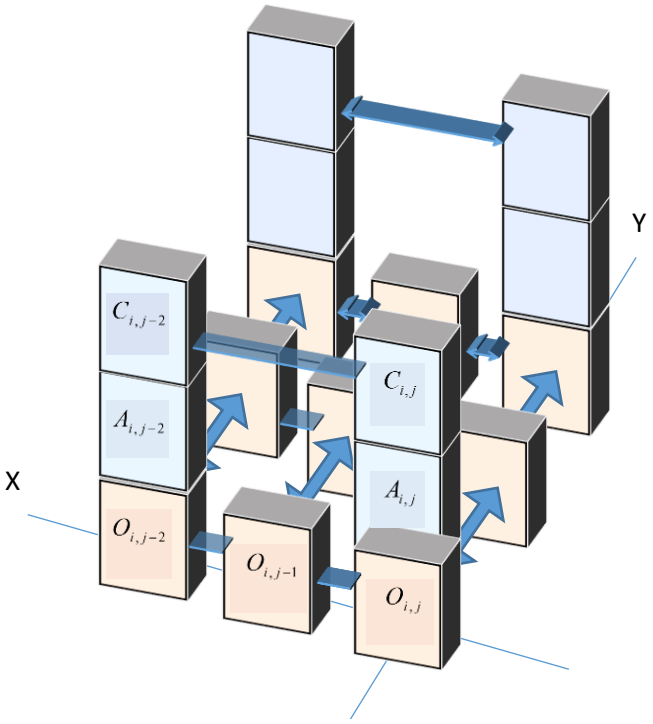


Рисунок 2.1 – Схема керування пласким об’єктом

У даній роботі поставлена задача розробити алгоритм і сервісну функцію, яка отримує координати точок керування розподіленим об’єктом (x, y, z) і для кожної точки керування знаходить усі сусідні, для яких виконується умова:

$$\frac{f(x,y,z)}{f(x_0,y_0,z_0)} \geq \varepsilon \quad (2.1)$$

де $f(x_0,y_0,z_0)$ – задане значення;

ε – задана точність

Також задається функція за якою буде поширюватися керування – $g(\Delta)$, де

$$\Delta = \sqrt{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2} \quad (2.2)$$

Очевидно, що

$$f(x,y,z) = g(\Delta) * f(x_0,y_0,z_0) \quad (2.3)$$

Виходячи із обчислених значень для кожної точки керування, будується матриця суміжності в якій «1» позначаються зв’язки між точками керування де виконується умова (2.1) та «0» ті зв’язки, де не виконується дана умова.

Узагальнений алгоритм знаходження сусідніх точок відображено на рисунку 2.2.

Виходячи із даної матриці потрібно побудувати та візуалізувати граф сусідства усіх точок керування.

Рисунок 2.2 – Блок-схема алгоритму знаходження сусідніх точок керування

2.2 Використання графу для відображення поширення впливу керування між точками керування

2.2.1 Визначення графу

Граф являє собою сукупність точок на площині або в просторі, і набір зв’язків між ними, що з'єднують дві точки або приєднується до самої себе. Граф G = (V (G), E (G)), складається з двох кінцевих точок. V (G) – набір вершин графа, позначається V, що являє собою не порожній набір елементів, званих вершинами та E (G) – набір ребер графа, часто позначається просто E, що є можливо порожнім набором елементів, які називаються ребрами [43]. Кожне ребро e в E має набір із однієї або двох вершин, пов'язаних з ним, які називаються кінцевими точками. Графічне відображення графа зображено на рисунку 2.3.

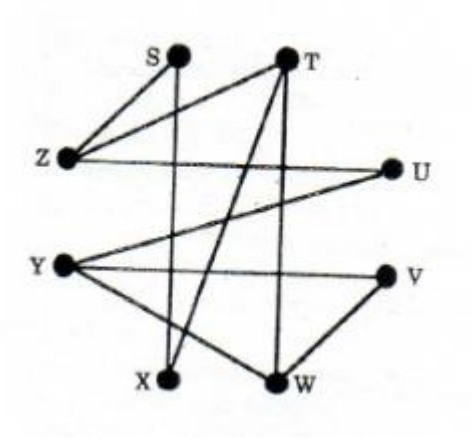


Рисунок 2.3 – Графічне відображення графа

Набір вершин V = {S, T, U, V, W, X, Y, Z}, набір ребер E має 10 ребер, і цим ребрам призначаються неупорядковані пари вершин:{(S, X), (S, Z), (T, W), (T, X), (T, Z), (U, Y), (U, Z), (V, W), (V, Y), (W, Y)}.

2.2.2 Застосування графів

Графи можуть бути використані для моделювання багатьох типів відносин і процесів у фізичних, біологічних, [44] соціальних та інформаційних системах. Багато практичних задач можуть бути представлені графами. Підкреслюючи їх застосування в реальних системах, термін мережа іноді визначається як граф, в якому атрибути (наприклад, імена) асоціюються з вузлами та / або ребрами.

У комп’ютерних науках графи використовуються для представлення мережевих зв'язків, організації даних, обчислювальних пристроїв, потоку обчислень тощо. Наприклад, структура посилань веб-сайту може бути представлена орієнтованим графом, у якому вершини представляють веб-сторінки і направлені ребра представляють посилання з однієї сторінки на іншу. Аналогічний підхід можна застосувати до проблем у соціальних медіа, подорожей, біології, дизайну комп'ютерних чіпів та багатьох інших сферах [45]. Тому розробка алгоритмів для обробки графів представляє великий інтерес у галузі інформатики. Перетворення графів часто формалізується і представлено системами перезапису графів. На додаток до систем перетворення графів, що фокусуються на керуванні графами на основі певних правил, є бази даних графів, орієнтовані на безпеку для транзакцій, стійкі зберігання та запити даних, структурованих за графами. Графотеоретичні методи в різних формах виявилися особливо корисними в лінгвістиці, оскільки природна мова часто добре піддається дискретній структурі. Традиційно синтаксис та композиційна семантика слідують за структурами на основі дерев, виразність яких полягає в принципі композиційності, змодельованому в ієрархічному графі.

Теорія графів використовується також для вивчення молекул у хімії та фізиці. У фізиці конденсованої речовини тривимірну структуру складних модельованих структур атомів можна кількісно вивчити, збираючи статистику щодо графотеоретичних властивостей, пов'язаних з топологією атомів. У хімії

граф робить природну модель для молекули, де вершини представляють зв'язки атомів та ребер. Цей підхід особливо застосовується при комп'ютерній обробці молекулярних структур, починаючи від хімічних редакторів до пошуку в базі даних. У статистичній фізиці графи можуть представляти локальні зв'язки між взаємодіючими частинами системи, а також динаміку фізичного процесу в таких системах. Аналогічно, в обчислювальній нейронауці графи можуть бути використані для представлення функціональних зв'язків між областями мозку, які взаємодіють, викликаючи різні когнітивні процеси, де вершини представляють різні ділянки мозку, а краї представляють зв'язки між цими областями [46].

У математиці графи корисні в геометрії та певних частин топології, таких як теорія вузлів. Теорія алгебраїчних графів тісно пов'язана з теорією груп. Структуру графа можна розширити, призначивши вагу кожному ребру графа. Графи з вагами або зважені графи використовуються для представлення структур, у яких попарні з'єднання мають деякі числові значення. Наприклад, якщо граф представляє дорожню мережу, ваги можуть представляти протяжність кожної дороги.

2.2.3 Матричне відображення графів

Графи можна відображати за допомогою різноманітних матриць. Наведемо приклади двох основних – матриці суміжності та матриці інцедентності.

Матриця суміжності графа $G = (V, E)$ – це матриця $n \times n$, $D = (d_{ij})$, де n – кількість вершин у G, $V = \{v_1, \ldots, v_n\}$ і d_{ij} = кількість ребер між v_i і v_j. Зокрема, $d_{ij} = 0$, якщо (v_i, v_j) не є ребром у G. Матриця D симетрична, тобто $D^T = D$ [47].

На рисунку 2.4 наведено приклад графа і його матриці суміжності.

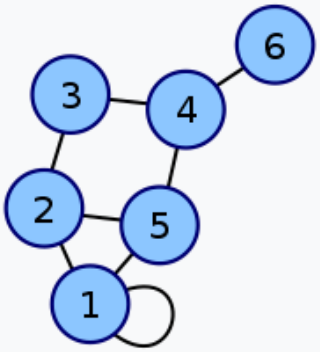
Маркований граф	Матриця суміжності
	$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ Координати 1-6.

Рисунок 2.4 – Приклад графа та його матриці суміжності

Матриця інцедентності всіх вершин непустого графа, що не містить циклів. Граф G = (V, E), є матрицею n × m. A = (a_{ij}), де n – кількість вершин у G, m – кількість ребер в G та $a_{ij} = \begin{cases} 1, \text{якщо } v_i \text{ кінцева вершина } e_j \\ 0, \text{ інакше} \end{cases}$ [48].

Приклад такої матриці зображено на рисунку 2.5.

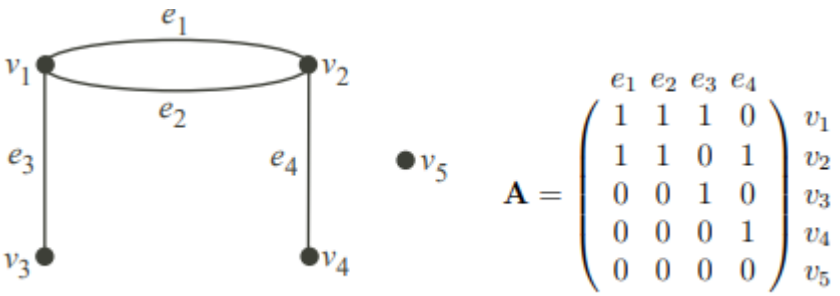
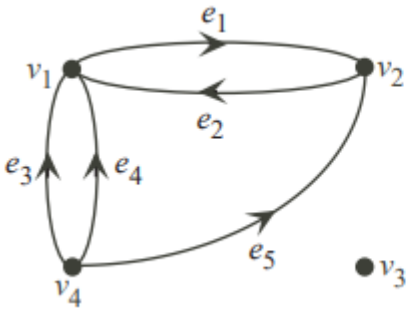


Рисунок 2.5 – Приклад матриці інцидентності у неорієнтованому графі

Матриця інцедентності всіх вершин непустого орієнтованого графа G, що не містить циклів: A = (a_{ij}), де $a_{ij} = \begin{cases} 1, \text{якщо } v_i \text{ початкова вершина } e_j \\ -1, \text{якщо } v_i \text{ кінцева вершина } e_j \\ 0, \text{ інакше} \end{cases}$

На рисунку 2.6 наведено приклад орієнтованого графа і його матриці інцедентності.



$$\mathbf{A} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 \end{matrix} \\ \begin{pmatrix} 1 & -1 & -1 & -1 & 0 \\ -1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{matrix}$$

Рисунок 2.6 – Приклад орієнтованого графа і його матриці інцедентності

Отже, в розробленій програмі побудований граф сусідства точок керування буде відображатися як графічно, так і за допомогою матриці суміжності.

2.3 Розробка алгоритму оптимізації, що базується на алгоритмі хвильового проходження вершин графа

Для проведення оптимізації в одноранговій системі управління, потрібно розробити алгоритм хвильового проходження через усі вершини графа у якому важливою складовою є порядок проходження цих вершин, які в свою чергу, є вузлами управління координаційною системою. Тому, для розробки даного алгоритму необхідно використати один із варіантів алгоритмів проходження через усі вершини графу.

2.3.1. Алгоритм пошуку в глибину

Пошук в глибину (DFS) – метод обходу дерева або графа. DFS проходить якомога глибше по одним шляхом, перш ніж створити резервну копію та спробувати інший [49].

Пошук в глибину можна порівняти із проходженням лабіринту. Рухаєтеся однією стежкою, і якщо потрапляєте в глухий кут, повертаєтесь назад і випробуєте іншу.

DFS починає обхід графа, відвідавши довільну вершину та позначивши її як відвідувану. При кожній ітерації алгоритм переходить до нерозглянутої вершини, яка примикає до тієї, в якій він перебуває. (Якщо таких вершин декілька, зв'язок можна вибрати довільно.) Цей процес триває до тих пір, поки не зустрінеться глухий кут – вершина, що не має суміжних невідвіданих вершин.

У глухому куті алгоритм створює резервне копіювання шляху до вершини, з якої він прийшов, і намагається продовжувати відвідувати невідвідані вершини звідти. Алгоритм врешті-решт зупиняється після резервного копіювання шляху до початкової вершини, яка є тупиком. На той час усі вершини в тому ж сполученому компоненті, що і початкова вершина, були відвідані. Якщо невідвідані вершини все ще залишаються, пошук в глибину повинен бути перезапущений на будь-якій з них.

Схему роботи алгоритму зображено на рисунку 2.7.

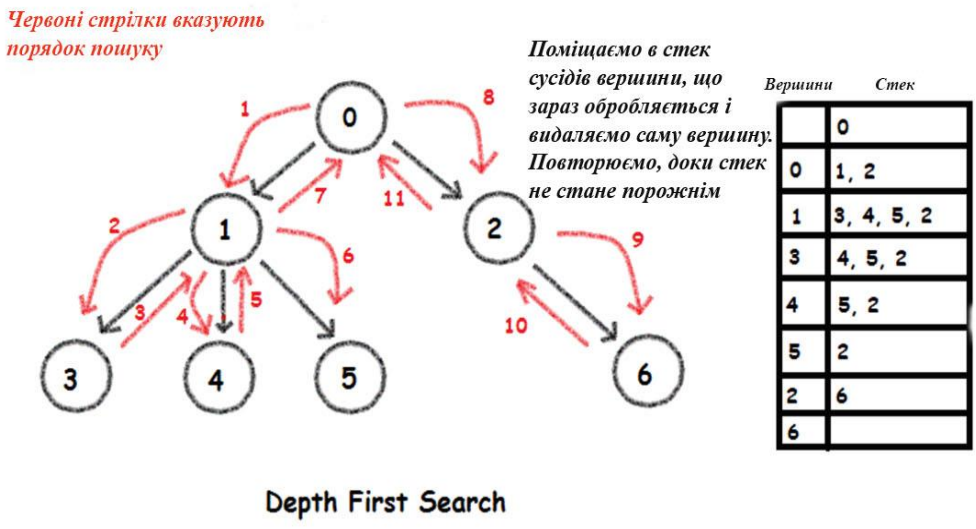


Рисунок 2.7 – Схема роботи алгоритму DFS

Кроки алгоритму:

1. Ініціюється порожній стек для розміру загальної кількості вузлів, S.

2. Для кожної вершини V задається, що V.visited = false.

3. Поміщаємо перший вузол, який потрібно відвідати, в Stack S.

4. Поки S не порожній:

Поміщаємо перший елемент у S, V.

Якщо V.visited = false, то:

V.visited = true

для кожної невідвіданої сусідньої вершини w із V:

Поміщаємо w у S.

5. Закінчується процес, коли всі вузли були відвідані [50].

Застосування DFS:

1. Для незваженого графа DFS обхід графа створює мінімальне остомне дерево. DFS лежить в основі алгоритмів Прима і Крускала.

2. Виявлення циклів у графі

Граф має цикл, тоді і лише тоді, коли ми бачимо зворотнє ребро під час DFS. Таким чином, ми можемо запустити DFS для графа і перевірити наявність зворотніх ребер.

3. Знаходження шляху між двома заданими вершинами.

4. Топологічне сортування

DFS є проміжним кроком для топологічного сортування.

5. Пошук сильно зв’язних компонентів графа. Орієнтований граф називається сильно зв’язним, якщо в графі є шлях від кожної вершини до кожної іншої вершини.

6. DFS дуже допомагає у вирішенні майже всіх задач проходження лабіринтів. Більшість головоломок лабіринту можна привести до задач із теорії графів, а результати обходу – у рішення. DFS можна адаптувати для пошуку всіх рішень для лабіринту, лише включивши вузли на поточний шлях у набір відвіданих вершин [51].

2.3.2 Пошук в ширину

Алгоритм пошуку в ширину - це техніка проходження графу, при якій вибирається випадковий початковий вузол (джерело або кореневий вузол) і починають обходити шари графа таким чином, щоб усі вузли та їх відповідні дочірні вузли були відвідані та досліджені [52].

При обході графа в ширину використовуються два поняття:

1. Відвідування вузла: так само, як підказує ім’я, відвідування вузла означає відвідувати або вибрати вузол.

2. Дослідження вузла: Вивчення сусідніх вузлів (дочірніх вузлів) вибраного вузла.

Графічно цей процес відображено на рисунку 2.8

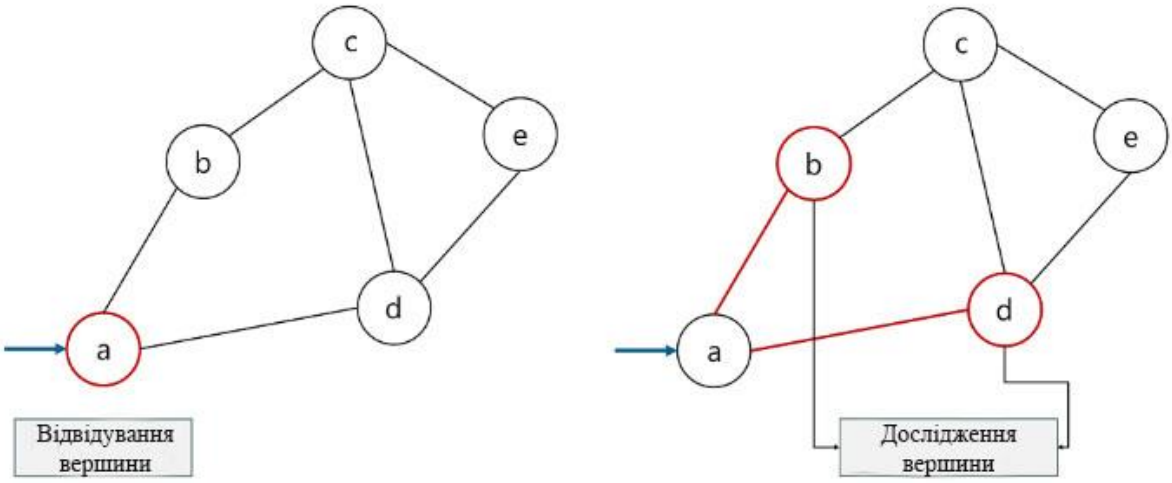


Рисунок 2.8 – Графічне відображення відвідування та дослідження вершин

Основною структурую даних при обході в ширину є черга, – це абстрактна структура даних, яка дотримується методології «Перший-ввійшов-перший вийшов» (до даних, що вводяться першими, можна отримати доступ спочатку) Він відкритий з обох кінців, де один кінець завжди використовується для вставки даних (enqueue), а другий використовується для видалення даних (dequeue). Графічно відображення черги зображено на рисунку 2.9.

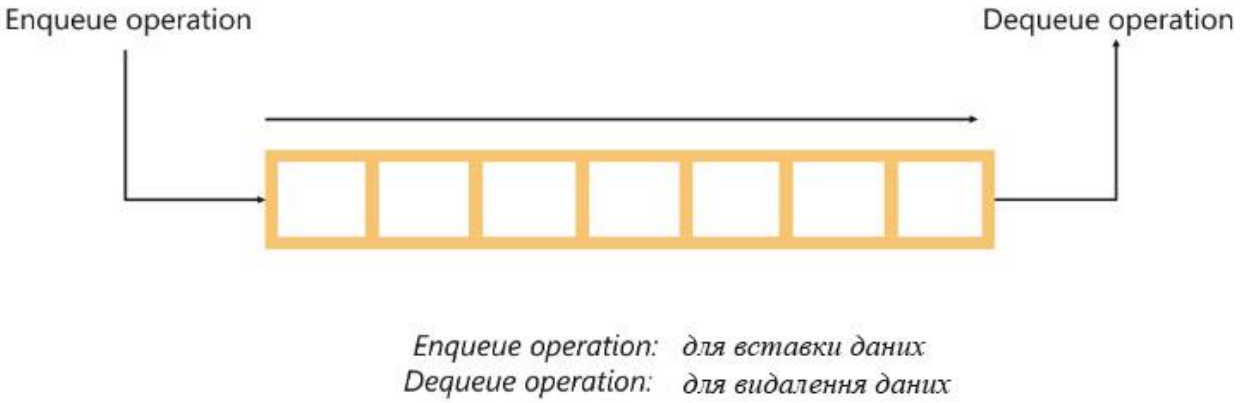


Рисунок 2.9 – Графічне відображення черги

Алгоритм пошуку в глибину для вирішення проблеми застосовує простий підхід, що базується на рівнях. Розглянемо нижче двійкове дерево (яке є графом). Мета полягає в тому, щоб обійти граф за допомогою алгоритму пошуку в ширину.

Розглянемо кроки, спрямовані на обхід графа за допомогою пошуку в ширину:

Крок 1: Створюється порожня черга.

Крок 2: Вибирається початковий вузол (відвідуваний вузол) і ставиться в чергу.

Крок 3: Якщо черга не порожня, вузол вилучається із черги та заносяться у чергу дочірні вузли (вузли, що досліджуються).

Крок 4: Друкується вузол, що вилучився.

Наведемо приклад використання алгоритму пошуку в ширину для графу показаного на рисунку 2.10.

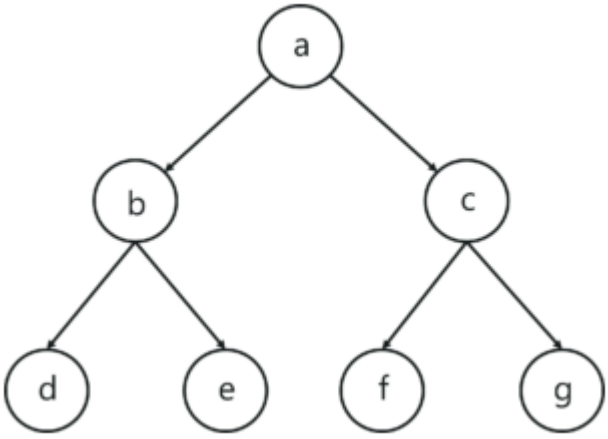


Рисунок 2.10 – Граф, що буде використаний для прикладу пошуку в ширину

Призначимо вузол 'a' як кореневий вузол і почнемо проходити вниз і виконувати вказані вище кроки.

Кінцевий процес проходження графа в ширину відображено на рисунку 2.11.

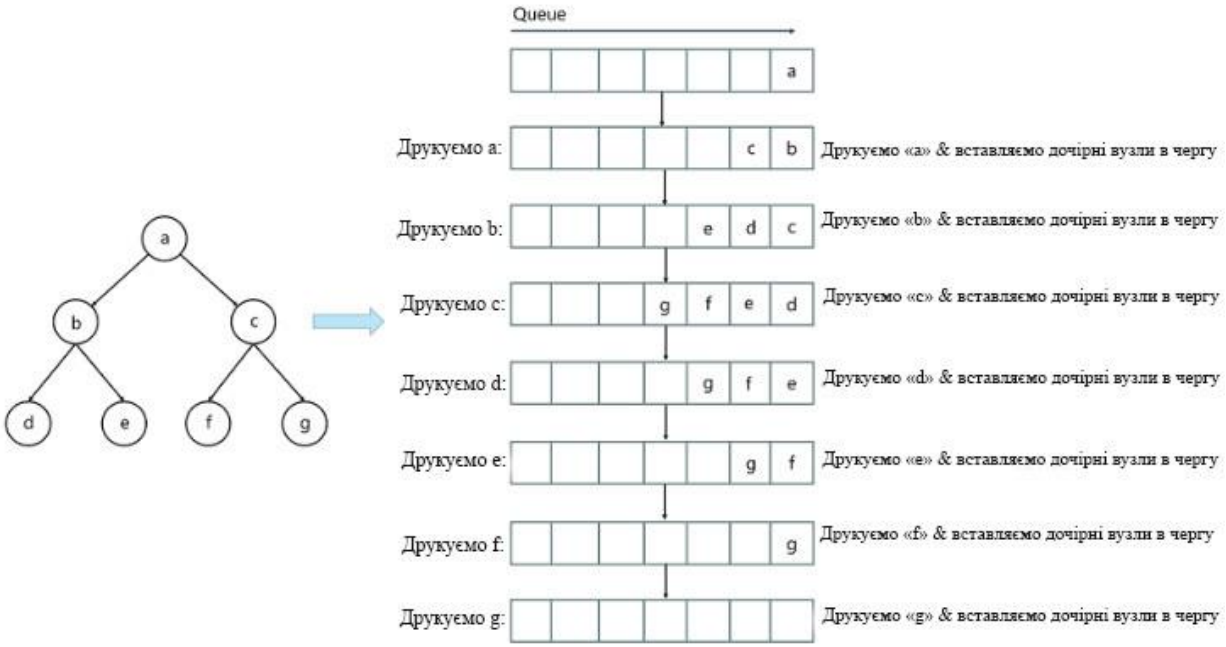


Рисунок 2.11 – Кінцевий процес проходження графа в ширину

Опишемо детальніше послідовність проходження графа(рис. 2.11):

1. Призначаємо 'a' як кореневий вузол та вставляємо його в чергу.

2. Вилучаємо з черги вузол 'a' та вставляємо дочірні вузли 'a', тобто 'b' та 'c'.

3. Друкуємо вузол 'a'

4. Черга не порожня і має вузли 'b' та 'c'. Оскільки 'b' є першим вузлом у черзі, вилучаємо його та вставляємо дочірні вузли 'b', тобто вузли 'd' та 'e'.

5.Повторюємо ці кроки, поки черга не порожня, враховуючи, що вже відвідані вузли не слід додавати до черги знову.

Застосування алгоритму пошуку в ширину:

Шукачі в пошукових системах: пошук в ширину – один з основних алгоритмів, що застосовуються для індексації веб-сторінок. Алгоритм починає переходити зі сторінки-джерела і слідкує за всіма посиланнями, пов’язаними зі сторінкою. Тут кожна веб-сторінка буде вважатися вузлом у графі.

Системи навігації GPS: пошук в ширину – один з найкращих алгоритмів, що використовуються для пошуку сусідніх локацій за допомогою системи GPS.

Знаходження найкоротшого шляху і мінімального остомного дерева для незважаного графа. Якщо мова йде про незважені графи, обчислити найкоротший шлях досить просто, оскільки процес вибору найкоротшого шляху буде полягати у виборі шляху з найменшою кількістю ребер. Пошук в ширину може дозволити це шляхом проходження мінімальної кількості вузлів, починаючи від вузла-джерела. Аналогічно, для знаходження остомного дерева, ми можемо використовувати будь-який із двох алгоритмів: пошуку в ширину чи пошуку в глибину.

Трансляція: Мережа використовує те, що ми називаємо пакетами для спілкування. Ці пакети дотримуються методу проходження, щоб дістатися до різних мережевих вузлів. Один з найбільш часто використовуваних методів обходу – пошук в ширину. Він використовується як алгоритм для передачі трансляційних пакетів по всіх вузлах мережі.

Мережі "Peer to Peer": Пошук в ширину може бути використаний як метод проходження для пошуку всіх сусідніх вузлів у мережі Peer to Peer. Наприклад, BitTorrent використовує Breadth-First Search для однорангового зв'язку [53].

Отже можна зробити висновок, що алгоритм пошуку в ширину є більш підходящим для хвильового проходження вершин графа при виконанні оптимізації, оскільки він проходить граф шар за шаром, має кращий час відгуку та при обробці великої кількості вузлів потребує менше ресурсів аніж рекурсивні алгоритми.

Обхід графа в ширину дасть порядок проходження вершин графа, який далі буде використаний у хвильовому алгоритмі при здійсненні процесу оптимізації.

2.3.3 Оптимізація методом Монте-Карло

Маючи однорівневу систему координаційного управління, що складається із n точок управління потрібно розрахувати максимальне y_{out} на кінцевій точці управління за допомогою оптимізації методом Монте-Карло, де

$$y_{out} = \beta \cdot x \tag{2.4}$$

$$x = \sum_{k=1}^{n} \left\{ y_{out\,k} \frac{\beta_k}{a \cdot t_k^{3/2}} e^{-\frac{r_k^2}{b \cdot t_k}} \right\} \tag{2.5}$$

y_{out}- вихід вершини, яка розглядається;

$y_{out\,k}$ - вихід k-ї сусідньої вершини;

β_k - це β, k-ї сусідньої вершини;

a, b - коефіцієнти

n - кількість сусідніх вершин;

r_k - «відстань» від k-ї сусідньої вершини. Це можна вважати вагою ребер графа

t_k - час розповсюдження впливу від k-ї сусідньої вершини, $t_k = \frac{r_k}{z}$, де $z = 2$.

Коефіцієнт β обчислюється із стабілізуючої системи, що зображена на рисунку 2.12, де

$$d = \beta_0 - \beta \quad 2.6$$

$$V = \frac{T_1 * V_{\text{попереднє}} + d * \Delta t}{T_1 + \Delta t} \quad 2.7$$

$$\beta = V * k \quad 2.8$$

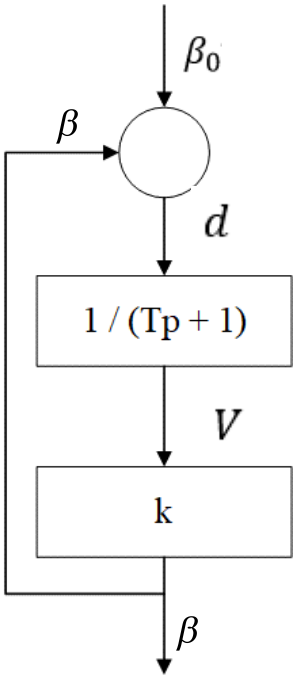


Рисунок 2.12 – Графічне відображення стабілізуючої системи

Методи Монте-Карло – це підмножина обчислювальних алгоритмів, які використовують процес повторювальної випадкової вибірки для чисельних оцінок невідомих параметрів [54].

Суть оптимізаційного процесу полягає в підборі таких вхідних β, при яких y_{out} буде максимальним. Вхідні β після початку оптимізації будуть почергово змінюватися в залежності від генератора випадкових чисел: збільшуватися на 10%, зменшуватися на 10%, або залишатися таким самим.

Так, як розробляється однорангова координаційна система, то підбір β буде відбуватися почергово для кожної вершини графа, послідовність вершин отримується за допомогою пошуку в ширину.

Отже, опишемо послідовність дій для алгоритму оптимізації, що базується на алгоритмі хвильового проходження вершин графа в режимі припинення обрахування по заданій кількості ітерацій:

1. Задаємо параметри: матрицю суміжності(adjMatrix), матрицю відстаней між вершинами(distanseMatrix), кількість ітерацій(iterationQnt), коефіцієнти(K, B, A, B0, alfa);

2. Знаходимо порядок проходження вершин алгоритмом пошуку в ширину;

3. Запускаємо цикл по кількості ітерацій;

4. По черзі для кожної вершини в порядку знайденої послідовності проходження вершин графа: генеруємо випадковий вплив на β, обраховуємо β за формулою (2.8), X за формулою (2.5), обраховуємо $У_{out}$ за формулою (2.4) для цієї вершини, обраховуємо кінцеве $У_{out}$ для всього графа($У_{out}$ на останній вершині), якщо його значення покращилося тоді залишаємо згенероване β, якщо погіршилося – повертаємо попереднє β;

5. Після того, як згенеровано β для всіх вершин порівнюємо $У_{out}$ із найкращим результатом і якщо він гірший, перезаписуємо $У_{out}$ та β для кожної вершини і переходимо до наступної ітерації;

6. Після закінчення всіх ітерацій виводимо користувачу $У_{out}$ та β найкращого результату;

В режимі припинення обрахування по заданому відхиленню обрахування закінчуються після того як відсоток відхилення результату від попереднього на 3 ітераціях попідряд попадає в заданий відсоток відхилення, а отже не змінюється більше ніж на заданий відсоток.

2.4 Розробка UML діаграм

UML (Unified Modeling Language) – уніфікована мова моделювання, що використовує графічні позначення для створення абстрактної моделі системи. Однією із основних причин використання UML діаграм є обмін інформацією та комунікація між розробниками.

2.4.1 Розробка діаграми варіантів використання

В розроблюваному сервісі актором виступає користувач. Користувач може:

1. Вводити параметри для побудови графу– користувач вводить кількість координат, точність та самі координати точок керування.
2. Побудувати граф – відбувається процедура побудови графу в 3d та відображення матриці суміжності.
3. Вибирати режим зупинки оптимізації – користувач вибирає один із двох можливих режимів – зупинки за кількістю ітерацій або зупинки за бажаним результатом.
4. Вводити обов’язкові параметри оптимізації – користувачу потрібно заповнити усі поля параметрів оптимізації для старту процесу оптимізації.
5. Старт та відображення результатів оптимізації – користувачу виводиться найкращий результат та табличка параметрів для усіх точок керування однорівневою координаційною системою.

Розроблена UML діаграма варіантів використання зображена на рисунку 2.13.

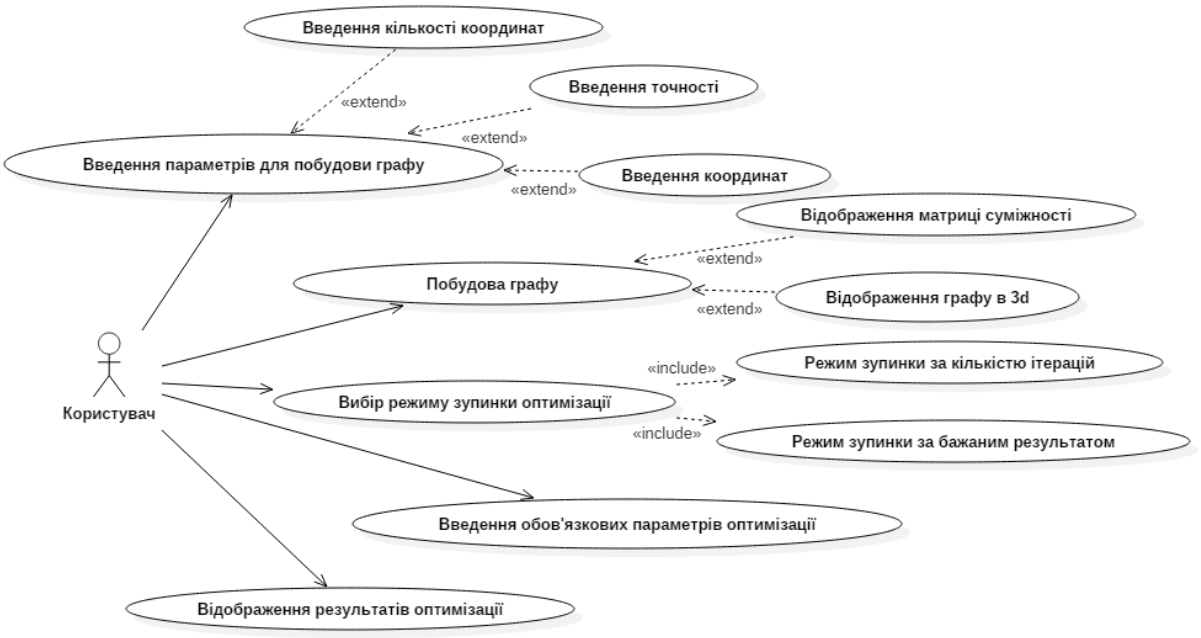


Рисунок 2.13 – Uml діаграма варіантів використання

2.4.2 UML-діаграма послідовності

UML-діаграма послідовності – діаграма послідовності показує впорядковане за часом спілкування між об'єктами. Ці діаграми відображають об'єкти системи та порядок відправлених повідомлень між ними.

Вертикальні лінії означають процеси або об'єкти, що існують водночас. Надіслані повідомлення – це горизонтальні ліній, відображені в порядку відправлення. Об’єкти можуть мати лінію життя, коли об’єкт видаляється, то нижня межа його лінії життя закінчується перехрестям, що є в даному випадку символом руйнування.

Розроблена UML діаграма послідовності зображена на рисунку 2.14.

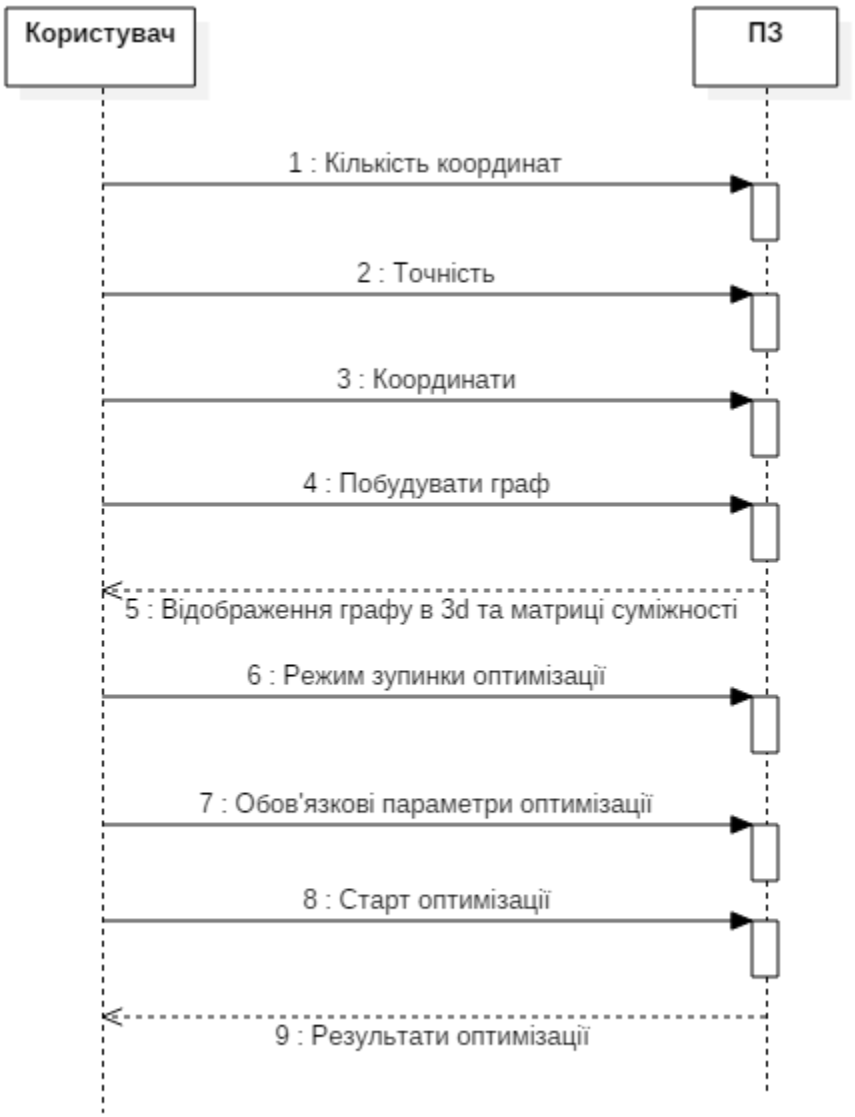


Рисунок 2.14 – UML-діаграма послідовності

З діаграми видно, що користувач починає взаємодіяти із системою шляхом введення вхідних даних необхідних для побудови графу шляхом використання алгоритму розробленого в розділі 2.1 даної роботи. Далі програма обраховує матрицю суміжності та будує граф в 3d. Користувач вибирає режим зупинки оптимізації та вводить обов’язкові параметри оптимізації. Потім користувач натискає кнопку розпочати оптимізацію за алгоритмом розробленим в розділі 2.3.3, після чого система виводить результати оптитимізації.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Мова програмування

Програма розроблена на мові програмування javascript. Javascript – мова програмування, що підтримує різні парадигми програмування, такі як об’єктно-орієнтовані, імперативні та функціональні стилі. Вона гарно підходить для виконання браузерних сценаріїв для надання інтерактивності веб сторінкам та виконання різних завдань на стороні клієнта.

Серед її переваг можна виділити:

1. Виконання на стороні клієнта. Виконання коду відбувається засобами процесора користувача, що зберігає пропускну спроможність та зменшує навантаження на сервер.

2. Javascript – легка у вивченні. Вона використовує модель DOM, яка надає великий набір готової функціональності для багатьох об’єктів на сторінці, що дозволяє легко розробляти скрипти для досягнення поставлених цілей.

3. Швидкий для кінцевого користувача. Так як код виконується на стороні клієнта результат отримується майже миттєво залежно від завдання, так як немає потреби обробки на стороні веб-сервера і відсилання назад результату, що зменшує локальну та пропускну здатність сервера.

4. Компіляція не потрібна, так як браузер інтерпретує Javascript як теги HTML.

5. Легкий в тестуванні та при дебагінгу.

6. Не залежний від платформи. Будь-який браузер із дозволеним Javascript може розуміти та інтерпретувати код. Будь-який Javascript код може бути виконаний на різних типах обладнання, для якого він написаний.

7. Наявні всі можливості процедурного програмування.

Серед недоліків можна виділити:

1. Javascript виконується на комп’ютерах користувача, що може бути використано в зловмисних цілях.

2. JavaScript виконується по-різному в різних браузерах, тоді як сценарії на стороні сервера завжди дають однаковий результат.

3.2 Огляд та тестування програмного забезпечення

При розробці програмного забезпечення однорівневою координаційною системою управління були використані алгоритми, описані у 2 розділі даної роботи, а саме алгоритм знаходження для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення, та хвильовий алгоритм проходження через усі вершини графа.

Для знаходження усіх сусідніх точок керування користувачеві потрібно ввести кількість точок керування та точність, що відображено на рисунку 3.1.

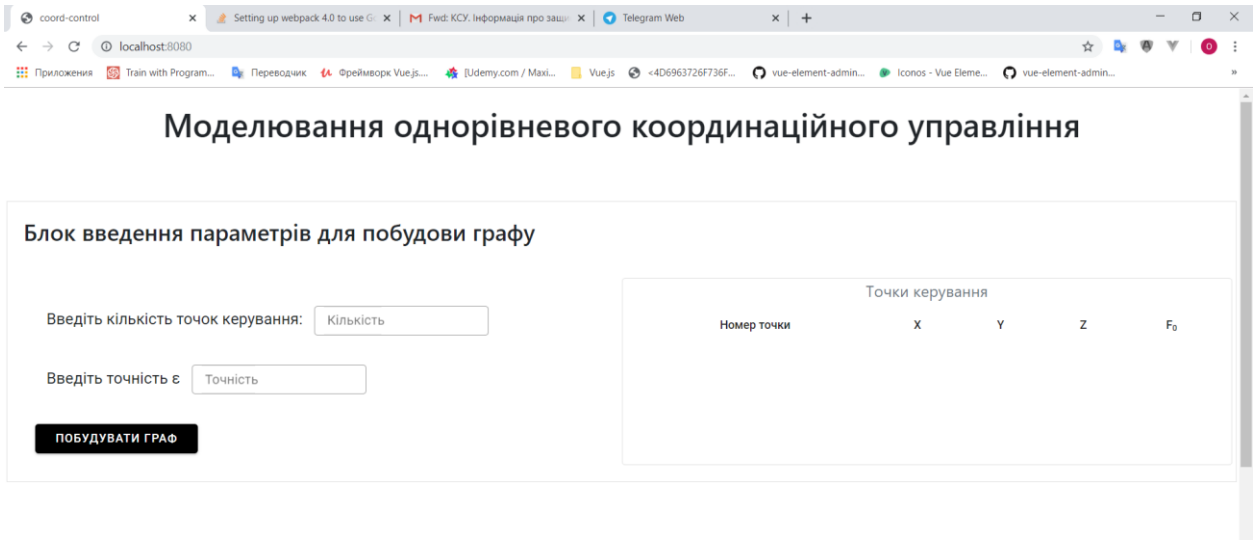


Рисунок 3.1 – Головне вікно програми

При зміні поля із кількістю точок керування автоматично в таблицю точок керування вноситься відповідна кількість пустих комірок для введення координат точок керування системою (рис 3.2)

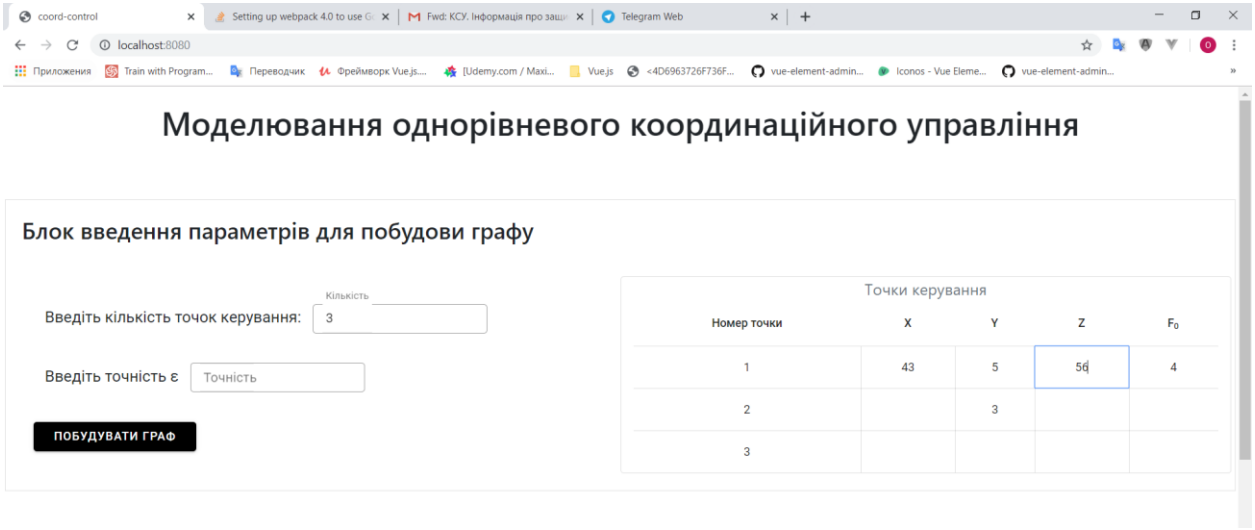


Рисунок 3.2 – Введення даних в таблицю координат точок керування

При введені координат система не дозволяє вводити ніяких інших знаків крім цифр та крапки для внесення десяткових дробів. При введені в поля із точністю та кількістю точок також є валідація, якщо значення не число – система інформує користувача про це виділенням червоним коліром поля, де введено помилкове значення (рис. 3.3). А в разі натиснення кнопки побудувати граф, – виведе повідомлення про не заповнені поля.

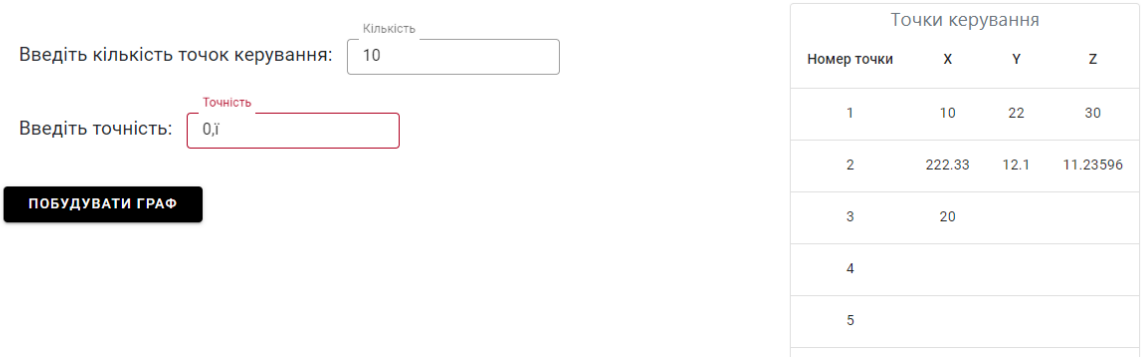


Рисунок 3.3 – Відображення помилкового значення

Після введення всіх необхідних даних і натиснення кнопки «Побудувати граф» система використовуючи алгоритм знаходження для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення знаходить ці точки

і відображає їх у вигляді матриці суміжності (рис. 3.4), та, так званого, графа сусідства (рис. 3.5).

Матриця суміжності

	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

Матриця відстаней

	1	2	3	4
1	0	2.83	3	2.24
2	2.83	0	2.24	4.12
3	3	2.24	0	2.83
4	2.24	4.12	2.83	0

Рисунок 3.5 – Матриці суміжності та відстаней між точками

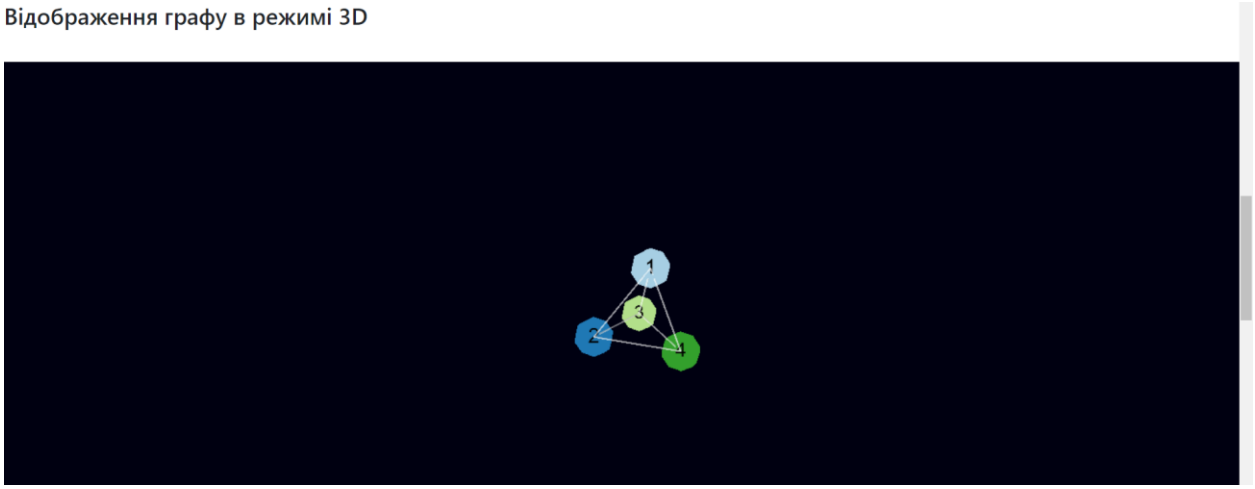


Рисунок 3.6 – Граф сусідства

Для розробки графа сусідства було використано бібліотеку three-forcegraph [55] та кастомізовано її під виконання задачі побудови графа сусідства точок керування. Граф відображається в 3d із можливістю перегляду із різних ракурсів (рис. 3.7).

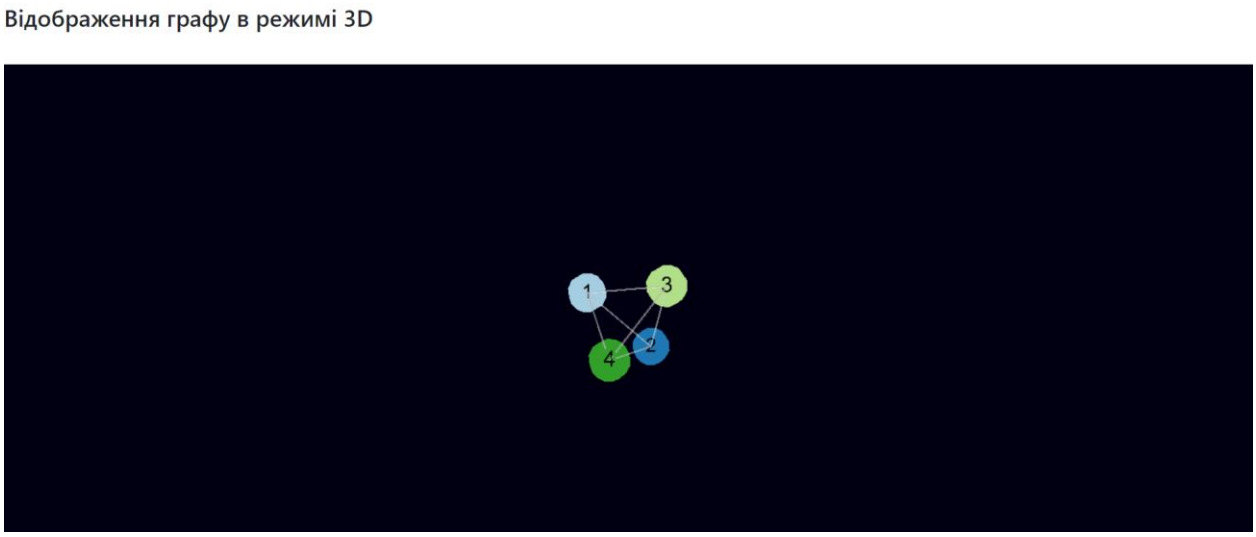


Рисунок 3.7 – Відображення графа із іншого ракурсу

Після побудови графу з’являється блок оптимізації (рис. 3.8) в якому є можливість вибору режиму зупинки оптимізації та блок введення параметрів оптимізації.

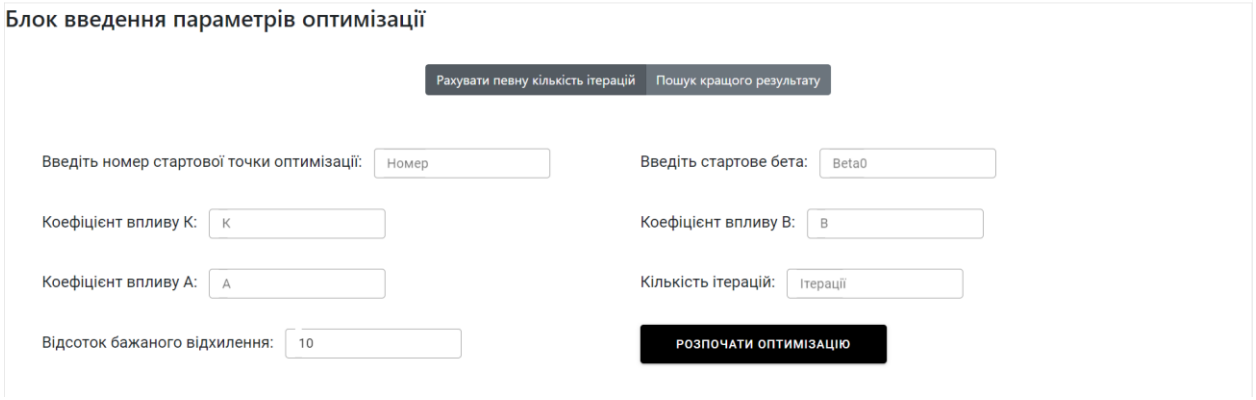


Рисунок 3.8 – Блок оптимізації

Для всіх полів введення також додана валідація із повідомленням користувачу про хибне введене значення.

При вибору режиму «Рахувати певну кількість ітерацій» використовується число введене в поле «Кількість ітерацій», а при режимі зупинки «Пошук кращого результату» – із поля «Відсоток бажаного відхилення».

Після заповнення всіх необхідних полів та натиснення кнопки «Розпочати оптимізацію» розпочинається процес оптимізації, після закінчення якого в табличку виводяться результати оптимізації, а саме β, β_0 та y_{out} для всіх вершин на найкращій ітерації. Значення y_{out} на останній вершині і буде найкращим результатом.

Наприклад для графу (рис. 3.9) при введених параметрах оптимізації (рис. 3.10) та режимі зупинки на кількість ітерації будуть такі результати(рис. 3.11). При таких же параметрах, але в режимі зупинки «Пошук кращого результату» результати оптимізації відображено на рисунку 3.12.

Матриця суміжності

	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

Матриця відстаней

	1	2	3	4
1	0	2.83	3	2.24
2	2.83	0	2.24	4.12
3	3	2.24	0	2.83
4	2.24	4.12	2.83	0

Рисунок 3.9 – Матриця суміжності

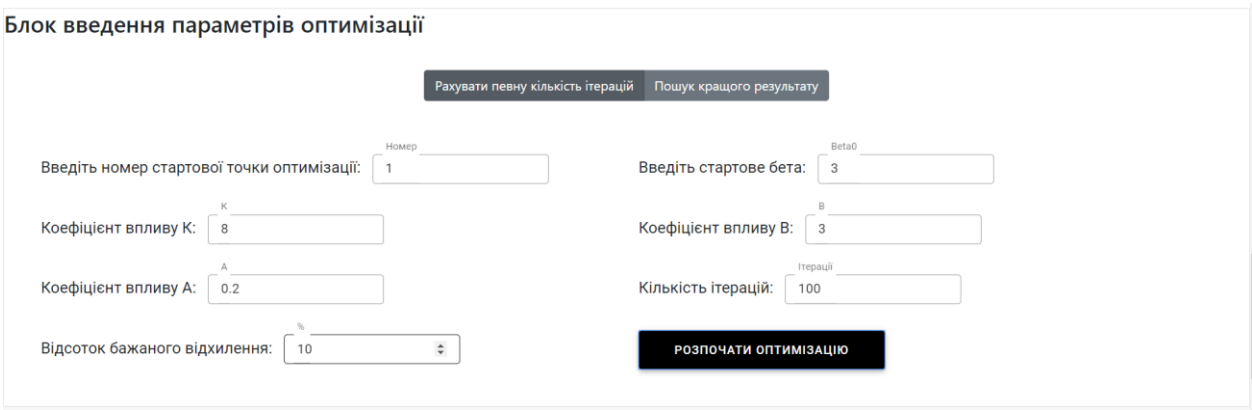


Рисунок 3.10 – Параметри оптимізації

РЕЗУЛЬТАТИ

Максимальний yOut : 14.405367342560188

	Beta	BetaZero	yOut
1	3.226666666666667	3.3000000000000003	17.048819363771
2	2.9333333333333336	3	12.60362853351961
3	2.64	2.7	14.227033123378556
4	3.156123215627348	3	14.405367342560188

Рисунок 3.11 – Результати оптимізації в режимі «Розрахунку певної кількості ітерацій»

РЕЗУЛЬТАТИ

Максимальний yOut : 17.898404644280294

	Beta	BetaZero	yOut
1	4.125465280173602	4.1770335961757725	24.48247621628821
2	4.251746400000002	4.348377000000002	19.915283283598754
3	2.6135999999999995	2.673	18.29806687418255
4	3.226666666666667	3.3000000000000003	17.898404644280294

Рисунок 3.12 – Результати оптимізації в режимі «Пошук кращого результату»

3.3 Супровідна документація

Для коректного використання програми користувачу необхідно мати інстукцію, у якій докладно пояснено послідовність дій для отримання бажаного результату. Також для можливого подальшого розширення функціоналу програмного забезпечення приведемо інструкцію програміста.

3.3.1 Інструкція користувача

Для роботи розробленого програмного забезпечення необхідно, щоб на ПК був встановлений браузер із доступом в мережу Інтернет, та у ньому було дозволено JavaScript.

Якщо попередня вимога виконана, то для роботи програми необхідно, зайти в папку dist та відкрити в браузері файл index.html.

В результаті в браузері відкриється сторінка введення параметрів та координат об’єкта керування (рис.3.12).

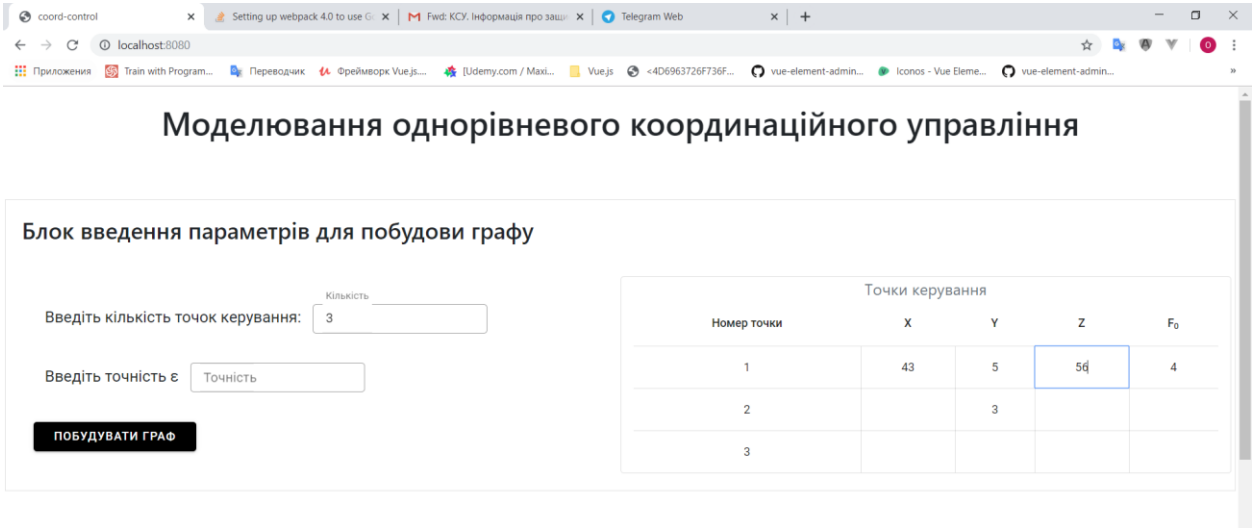


Рисунок 3.12 – Введення параметрів та координат

При введені параметрів система перевіряє правильність введених даних, та в разі помилки повідомить про неї. Після введеня параметрів потрібно натиснути кнопку «Побудувати граф». В результаті система побудує та відобразить граф(рис. 3.14) та його матрицю суміжності(рис. 3.13).

Рисунок 3.13 – Матриця суміжності та дистанцій

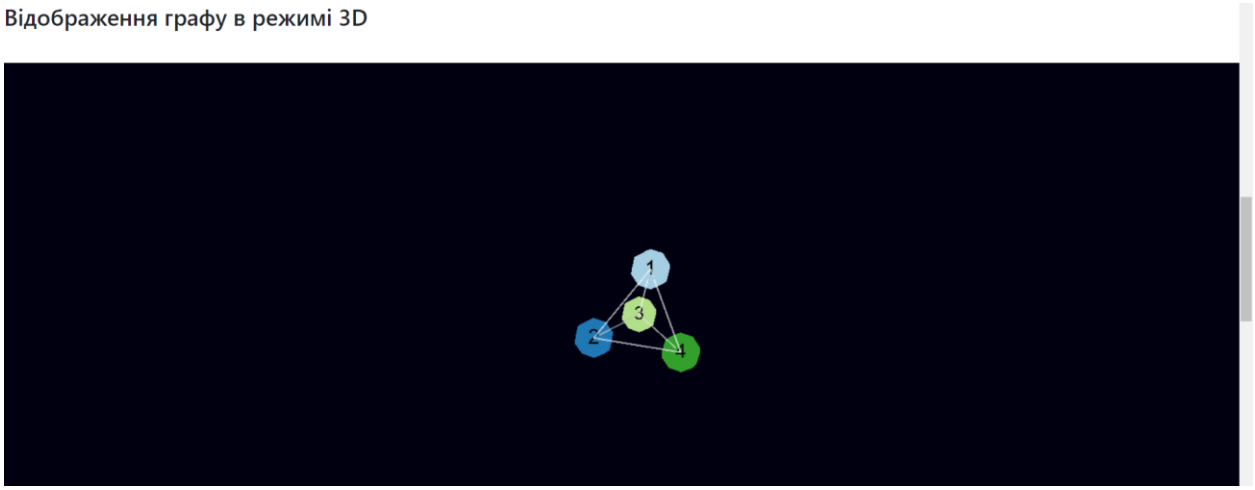


Рисунок 3.14 – Граф сусідства

Після чого необхідно вибрати режим зупинки оптимізації. Режим «Рахувати певну кількість ітерацій» потребує вказання кількості ітерацій, і в результаті виведе результат на найкращій ітерації. В свою чергу режим «Пошук кращого результату» потребує вказання відсотку бажаного відхилення, і в результаті зупинить процес оптимізації, якщо відсоток відхилення на протяжності трьох ітерацій буде змінюватися в межах заданого відхилення і виведе найкращий результат серед проведених операцій.

Після вибору режиму зупинки потрібно ввести параметри оптимізації(рис.3.15) та натиснути кнопку «Розпочати оптимізацію». Результати оптимізації виводяться у табличку (рис. 3.16).

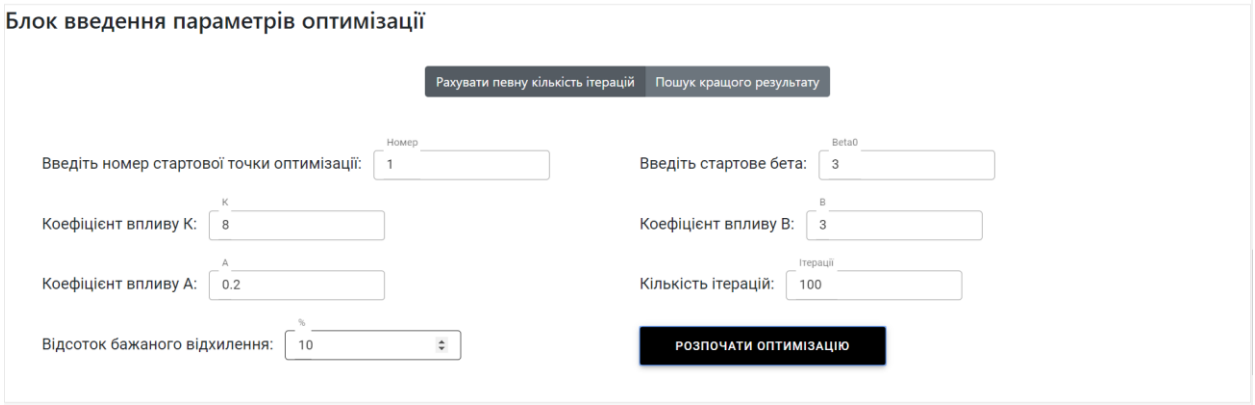


Рисунок 3.15 – Параметри оптимізації

РЕЗУЛЬТАТИ

Максимальний yOut : 17.898404644280294

	Beta	BetaZero	yOut
1	4.125465280173602	4.1770335961757725	24.48247621628821
2	4.251746400000002	4.348377000000002	19.915283283598754
3	2.6135999999999995	2.673	18.29806687418255
4	3.226666666666667	3.3000000000000003	17.898404644280294

Рисунок 3.16 – Результати оптимізації

3.3.2 Інструкція програміста

Для відкриття вихідного коду програми необхідно встановити будь-який редактор коду із підримкою мови javascript. Проте серед найбільш зручних, легких у використанні, багатофункціональних та до того ж безкоштовним є Visual Studio Code.

Також потрібно щоб на ПК було встановлено середовище виконання javascript node v10.15.3 і вище, а також менеджер пакетів npm 6.4.1 і вище.

Проект створеного програмного забезпечення зображено на рисунку 3.17.

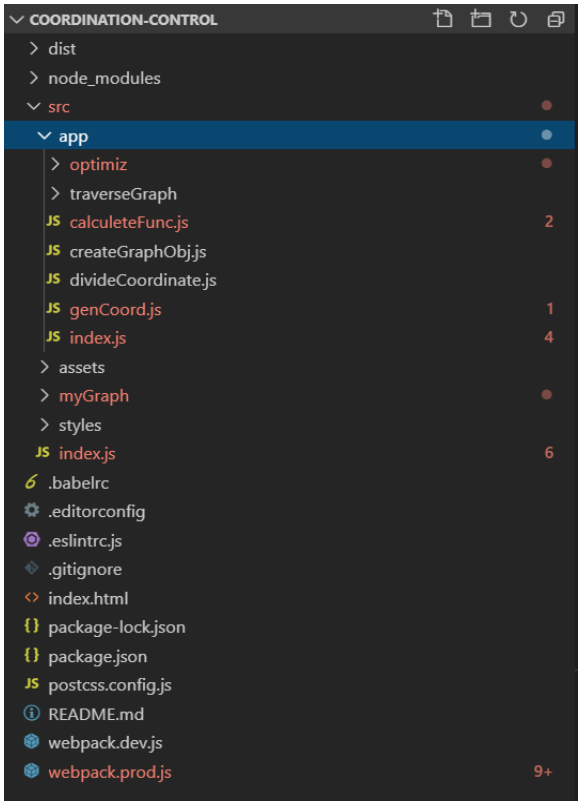


Рисунок 3.17 – Проект створеного програмного забезпечення

Файл index.html – вхідна точка веб-сервісу, саме в ньому розміщуються елементи DOM та підключаються скрипти.

Скрипт src/index.js – збирає докупи усі модулі, реалізована логіка роботи із DOM елементами та валідація вхідних даних

Скрипт src/app/index.js відповідає за алгоритм знаходження для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення

Скрипт src/app/traverseGraph/three.js реалізовує алгоритм обходу графа в ширину.

Скрипт src/app/optimiz/main.js – здійснює реалізацію оптимізації за допомогою хвильового алгоритму.

Для початку роботи потрібно зайти в терміналі в папку проекту і виконати ряд команд:

npm install – встановить всі необхідні залежності для проекта.

npm start –запуск проекту в режимі розробника.

Команда npm run build збере production версію проекта, де будуть лише необхідні залежності для кінцевого користувача, а не для розробника.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки програмного забезпечення системи координаційнного управління, створеного в результаті науково-технічної діяльності.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету, кафедри комп'ютерних систем управління: д.т.н., проф. Дубового В. М., доцента, к.т.н., Юхимчука М. С., доцент к.т.н Ковтуна В. В. За допомогою таблиці 4.1 за п'ятибальною шкалою використовуючи 12 критеріїв оцінки комерційного потенціалу розробки експерти надали свої оцінки.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження табл. 4.1

5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Продовження табл. 4.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 4.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 4.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Дубовой В. М.	Юхимчук М. С.	Ковтун В. В.
	Бали, виставлені експертами:		
1	3	3	4
2	4	4	3
3	2	3	2
4	3	4	3
5	2	2	2
6	2	1	3
7	1	2	1

Продовження табл. 4.3

8	2	3	0
9	2	3	1
10	4	4	4
11	3	4	4
12	4	3	4
Сума балів	$СБ_1$=32	$СБ_2$=36	$СБ_3$=31
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{32+36+31}{3} = 33$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 33, що згідно таблиці 4.2 вважається, що рівень комерційного потенціалу розробки є вище середнього.

Розробка буде реалізуватися на кафедрі КСУ, для здійснення моделювання однорівневих систем управління і використовуватиметься вона як студентами так і викладачами.

В якості аналога для розробки було обрано Condor – система для підтримки і моделювання середовища High Throughput Computing (HTC).

Основними недоліками аналога є: офіційно не підтримуються безкоштовні операційні системи, не зрозумілий та не дружній інтерфейс користувача. Також до недоліків можна віднести особливості та труднощі, що виникають при запуску задач під операційною системою Windows.

У розробці дана проблема не виникає так як розроблена система є кросплатформеною. За рахунок кросплатформеності система запускається в будь-якій сучасній операційній системі без різного роду ускладнень та помилок. Також враховано складність інтерфейсу для користувача, що наявний в конкурента і запропоновано власний, інтуїтивний та зручний для користувачів, що підтвердило проведене тестування системи.

Також система випереджає аналог за такими параметрами як відображення результатів моделювання, що наочно демонструються в вигляді побудови 3d графа сусідства точок керування.

В таблиці 4.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 4.4 - Основні технічні показники аналога і нової розробки

Показники	Аналог	Нова розробка	Відношення параметрів нової розробки до параметрів аналога
Знаходження для кожної із точок керування таких сусідніх точок, на які поширюється вплив згідно із заданим законом поширення	Ні	Так	-
Побудова графу сусідства точок керування	Ні	Так	-
Мова інтерфейсу	Англійська	Українська	-
Швидкісь розрахунку(із однаковими вхідними даними)	9,8с	9,6с	0,98
Виведення результатів	Екран	Екран	-

Виходячи із даних таблиці 4.4 можна зробити висновок, що аналог має гірші технічні показники ніж розроблене програмне забезпечення, яке має кращу швидкодію(перевірено за однакових умов) та більшу функціональність. Це свідчить про те, що розроблена система є зручнішою і надає більші можливості користувачу, не поступаючись швидкодією.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Проведемо прогнозування витрат на виконання робіт у три етапи:

*1-й етап:*Розрахунок витрат, які безпосередньо стосуються виконавців даного розділу НДР.

2-й етап: Розрахунок загальних витрат на виконання НДР;

3-й етап: Прогнозування загальних витрат на виконання та впровадження НДР.

Проведемо перший етап прогнозування.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_О = \frac{М}{Т_р} * t \text{ (грн)} \quad (4.1)$$

де $М$ – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_Р$ – число робочих днів в місяці; приблизно $Т_Р \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

Дану розробку буде проводити програміст, величина окладу буде становити 6000 грн. на місяць. Кількість робочих днів у місяці складає 22, а кількість робочих днів дослідника складає 45. Зведемо сумарні розрахунки до таблиця 4.5.

Таблиця 4.5 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник проекту	11000	500.0	5	2500
Програміст	6000	272.7	45	12273
Всього				14773

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників. На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = 0{,}1 * 14773 = 1625 \text{ (грн)} \quad (4.2)$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (4.3):

$$Н_{ЗП} = (З_о + З_д) * \frac{\beta}{100} \text{ (грн)} \quad (4.3)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді**:**

$$Н_{ЗП} = (14773 + 1625) * \frac{22}{100} = 3607{,}5 \text{ (грн)}$$

4. Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$A = \frac{Ц \cdot T}{Ткор \cdot 12} \quad [грн], \tag{4.4}$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункта 137.3.3 Податкового кодекса амортизація нараховується на основні засоби вартістю понад 2500 грн.

Всі проведені розрахунки амортизаційних відрахувань заносимо в табл. 4.6.

Таблиця 4.6 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень	Балансова вартість, грн.	$t_{кор}$ (р)	Термін використання міс.	Величина амортизаційних відрахувань, грн.
Комп'ютер	10000	2	1	416,67
Приміщення	90000	20	1	375,00
Всього				791,67

5. Норма витрат матеріалу – це плановий показник, який визначає максимально допустимі затрати відповідних ресурсів на виробництво одиниці продукції в умовах певного рівня техніки і організації виробництва.

Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \quad \text{грн.}, \tag{4.5}$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг;

$Ц_i$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

$В_i$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Інформацію про використані матеріали подамо у вигляді табл. 4.7.

Таблиця 4.7– Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
папір	100	1	100
ручка	15	1	15
флешка	120	1	120
З врахуванням коефіцієнта транспортування			258.5

6. До статті «Паливо та енергія на технологічні цілі» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються у процесі виробництва продукції. У даному випадку будемо враховувати лише витрати на електроенергію, яку споживає комп'ютер:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (4.6)$$

де В – вартість 1 кВт енергії, грн. В = 8,44 грн/кВт*год;

П – установлена потужність обладнання, кВт. П = 500 Вт або П = 0,5 кВт;

Ф – фактична кількість годин роботи обладнання, год. Ф = 100 год.;

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,65.

$$В_е = 8{,}44 \cdot 0{,}5 \cdot 100 \cdot 0{,}65 = 274{,}3 (грн).$$

Інші витрати $B_{ін}$ охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Інші витрати $B_{ін}$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$B_{ін} = (1..3) \cdot (З + З_р). \quad (4.7)$$

$$B_{ін} = 1 \cdot 14773 = 14773 (грн.)$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$B = 14773 + 1625 + 3607,5 + 791,67 + 258,5 + 274,3 + 14773 = 36102,4 (грн.)$$

Загальна вартість всієї МКНР визначається за формулою:

$$B_{заг} = \frac{B}{\alpha} \qquad (4.8)$$

$$B_{заг} = \frac{36102,4}{0,9} = 40113,8 (грн.)$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{B}{\beta}, \qquad (4.9)$$

де $\beta -$ коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт β = 0,1.

Звідси:

$$ЗВ = \frac{40113,8}{0,1} = 401138 (грн.).$$

4.3 Оцінка внеску НДР

Для цього спочатку потрібно визначити коефіцієнт наукової значимості отриманих результатів НДР, а потім підрахувати внесок в досягнення цих результатів.

Коефіцієнт наукової значимості результатів проведеної науково-дослідної роботи $К_{ЗН}$ можна підрахувати за формулою:

$$К_{ЗН} = \frac{\sum_1^3 b_i * d_i}{\sum_1^3 b_{max} * d_i} \quad (4.10)$$

де b_i – значимість отриманих результатів: b_1 – ступінь наукової новизни, b_2 – рівень теоретичної обґрунтованості, b_3 – ступінь експериментальної перевірки результатів. Бальна оцінка отриманих результатів наведена в таблиці 4.8. Максимальне значення отриманих за кожною характеристикою результатів можна прийняти в межах 7...10 балів;

d_i – питома вага кожної характеристики, значення якої наведені в таблиці 4.8;

З – кількість характеристик, за якими була зроблена оцінка результатів науково-дослідної роботи.

Таблиця 4.8 – Показники для оцінювання наукової значимості результатів виконаної НДР

Характеристики	Питома вага характеристики d_i	Бальна оцінка характеристики		
		Ступінь новизни b_1	Рівень теоретичної обґрунтованості b_2	Ступінь експериментальної перевірки результатів b_3
		1	2-5	7-10
b_1	0,500	Часткове удосконалення виробів, технологій, матеріалів, програмного продукту тощо	Суттєве удосконалення виробів, технологій, матеріалів, програмного продукту тощо	Нові напрямки в розробці виробів, технологій, матеріалів, програмного продукту тощо. Створення принципово нової техніки
b_2	0,333	Позитивне рішення на основі зроблених узагальнень	Установлення залежносте, які використовувались в інших випадках	Відкриття нових шляхів рішення задачі

Продовження табл. 4.8

b_3	0,167	Експериментальна перевірка не робилась	Результати перевірялись на невеликій кількості даних	Результати перевірені на великій кількості даних

В ході виконання даної магістерської кваліфікаційної роботи було розроблено програмне забезпечення системи координаційнного управління, створеного в результаті науково-технічної діяльності.

Тоді згідно таблиці 4.8 $b_1 = 4$. Отримані результати підтвердили залежності та характеристики, отримані раніше для окремих випадків рішення даної задачі, тому $b_2 = 4$. Оскільки результати перевірялись на отриманих попередньо експериментальних даних, приймаємо $b_3 = 5$. Тоді коефіцієнт наукової значимості результатів буде наступним

$$K_{ЗН} = \frac{4 \cdot 0,5 + 4 \cdot 0,333 + 5 \cdot 0,167}{10 \cdot 0,5 + 10 \cdot 0,333 + 10 \cdot 0,167} = 0,43.$$

Зважаючи на малий час, відведений на проведення дослідження, отримане значення коефіцієнта наукової значимості можна вважати цілком задовільним.

Внесок дипломника в досягнення отриманих результатів НДР можна розрахувати за формулою:

$$V = \frac{k_{\text{ТВІ}} * З_i}{\sum_i^n k_{\text{ТВІ}} * З_i} \quad (4.11)$$

де $k_{\text{ТВІ}}$ – коефіцієнт творчої участі кожного виконавця НДР, який оцінюється таким чином: проведення досліджень – 3 бали, робоче проектування – 1,5 бали, освоєння – 1,0 балів. якщо виконавець приймав участь в декількох видах робіт, то береться сума відповідних балів;

$З_i$ – заробітна плата кожного виконавця НДР, грн.;

n – кількість всіх виконавців науково-дослідної роботи: наукові керівники, відповідальні виконавці, спеціалісти, наукові співробітники, робітники різних професій тощо.

$$V = \frac{3 * 11000}{3 * 11000 + 6000 * 1{,}5} = 0{,}79$$

4.4 Висновок

В даному розділі було оцінено комерційний потенціал розробки і визначення коефіцієнта наукової значимості розробки програмного забезпечення системи координаційнного управління.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 36102,4 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 401138 грн.

Отримане значення коефіцієнта наукової значимості ($K_{ЗН} = 0,43$) можна вважати цілком задовільним, оскільки час відведений на проведення дослідження незначний. Особистий внесок дипломника в наукове дослідження становить 0,79, що являється нормальним значенням.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи розроблено програмне забезпечення координаційного управління однорівневою системою.

В першому розділі був проведений детальний аналіз існуючих однорівневих систем координаційного управління та розглянуто варіанти побудови архітектури систем, наведено їх недоліки та переваги.

Другий розділ присвячено розробці алгоритмів та uml діаграм програмного забезпечення однорівневого координаційного управління, також розроблено метод знаходження для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення. Наведено опис алгоритму обходу графу в ширину, та на його основі розроблено хвильовий алгоритм для проходження усіх вершин графа та здійснення оптимізації методом Монте Карло. Також в ньому розроблено та описано uml діаграми варіантів використання та послідовності.

В третьому розділі було описано розроблене програмне забезпечення та проведено його тестування. Тестування встановило, що розроблене програмне забезпечення успішно справляється із поставленою перед ним задачею та видає коректні результати. Також розроблено супровідну документацію для користувача та програміста.

Четвертий розділ присвячено економічному обґрунтуванню розробки. Під час розробки даного розділу було здійснено оцінку економічного потенціалу розробки. Спрогнозовано витрати на здійснення науково-дослідної та технологічно-конструкторської роботи. Також було отримано значення коефіцієнта наукової новизни та особистого внеску автора магістерської роботи.

Отже, можна зробити висновок, що розроблене програмне забезпечення можна використовувати для моделювання однорівневої координаційної системи управління.

СПИСОК ЛІТЕРАТУРИ

1. Глушец В. А., Руппель А. А. Проектирование автоматизированных систем: учебное пособие / В. А. Глушец, А. А. Руппель – Омск: Сибади, 2009. – 145 с.

2. Іванов А. О. Теорія автоматичного керування: Підручник / А. О. Іванов. – Дніпропетровськ: Національний гірничий університет, 2003. – 250 с.

3. Месарович М., Мако Д., Такахара И. Теория иерархических многоуровневых систем / М. Месарович, Д. Мако, И. Такахара – Москва: МИР, 1973. – 344с.

4. Peer-to-peer [Електронний ресурс] / Wikipedia –2018. – Режим доступу до ресурсу https://uk.wikipedia.org/wiki/Peer-to-peer

5. General info on FidoNet [Електронний ресурс] / Fidonet –2019. – Режим доступу до ресурсу https://www.fidonet.org/genlinfo.html

6. Min-JenTsai, Yin-Kai Hung Distributed computing power service coordination based on peer-to-peer grids architecture / Min-JenTsai, Yin-Kai Hung. – Expert Systems with Applications, 2009. – 17p.

7. .Napster media sharing system. [Електронний ресурс] / NAPSTER – 2018. – Режим доступу до ресурсу http://www.napster.com/.

8. The History of Napster [Електронний ресурс] / lifewire – 2019. – Режим доступу до ресурсу https://www.lifewire.com/history-of-napster-2438592

9. Alderman J. Sonic Boom: Napster, P2P and the Battle for the Future of Music / J.Alderman. – New York: Fourth Estate, 2014. – 234 с.

10. How the Old Napster Worked [Електронний ресурс] / Tech – 2019. – Режим доступу до ресурсу https://computer.howstuffworks.com/napster2.html

11. Gnutella P2P Free File Sharing and Download Network [Електронний ресурс] / lifewire – 2019. –Режим доступу до ресурсу https://www.lifewire.com/definition-of-gnutella-818024

12. Oram A. Peer-to-Peer / A. Oram – New York: O`Reilly Media, 2001. – 205 с.

13. Clarke I., Sandberg O., Wiley B., Hong T. W. Freenet: A Distributed Anonymous Information Storage and Retrieval System / I. Clarke, O. Sandberg, B. Wiley, T. W. Hong– Lecture Notes in Computer Science, 2000. – 20p.

14. P2P Networks Evolve: An Interview with StreamCast CEO Michael Weiss [Електронний ресурс] / technewsworld – 2004. –Режим доступу до ресурсу https://www.technewsworld.com/story/32673.html

15. Taylor Ian J., Harrison A. From P2P and Grids to Services on the Web: Evolving Distributed / Ian J. Taylor, A. Harrison – London: Springer, 2009. – 302 p.

16. Thurner S., Hanel R., Bo Liu, Corominas-Murtra B. Understanding Zipf's law of word frequencies through sample-space collapse in sentence formation / S. Thurner, R. Hanel, Bo Liu, B. Corominas-Murtra– Journal of the royal society interface, 2015. – 10 p.

17. Порєв Г. В. АРХІТЕКТУРНІ ЗАСАДИ РОЗПОДІЛЕНИХ ТА ОДНОРАНГОВИХ МЕРЕЖ / Г. В. Порєв – Київський політехнічний інститут, 2010. – 7 с.

18. Тригуб С. Н. Програма мережевої академії Cisco CCNA 1 и 2 / С. Н. Тригуб – Москва: Вильямс, 2005. – 1145 с.

19. Warland J., Varaiya P. High-Performance Communication Networks / J. Warland, P. Varaiya – California: Elsevier, 2000. – 693 p.

20. Tanenbaum A. S. Distributed Operating Systems / A. S. Tanenbaum – Prentice Hall, 1994. – 325 p.

21. Kahanwal B. The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle / B. Kahanwal, T. Pal Singh // International Journal of Latest Research in Science and Technology. –2012. – № 2. – p. 183-187

22. Coulouris G. Distributed Systems: Concepts and Design / Coulouris G., Dollimore J., Kindberg T. – New York: Addison-Wesley, 2001. – 1025 p.

23. Tanenbaum A. S., Steen M. Distributed Systems: Principles and Paradigms / A. S. Tanenbaum, M. Steen . – California: Pearson Prentice Hall. – 2007. – 686 p.

24. Ghosh S. Distributed Systems: An Algorithmic Approach / S. Ghosh – CRC Press: Dortmud, 2014. – 554 p.

25. Koubarakis M. Databases, Information Systems, and Peer-to-Peer Computing / M. Koubarakis, V. Kalogeraki, K. Aberer. – Berlin: First International Workshop. – 2003. – 621 p.

26. Foster I., Kesselman C. Globus: A Metacomputing Infrastructure Toolkit // International Journal of Supercomputer Applications Vol. 11, Issue 2. 1997. P. 115-128.

27. On Cheating, P2P, and [non-]Authoritative Servers from “D&D of MMOG” book [Електронний ресурс] / IT Hare– 2015. –Режим доступу до ресурсу http://ithare.com/chapter-iii-on-cheating-p2p-and-non-authoritative-servers-from-dd-of-mmog-book/

28. Стивенс Д., Камер Д. Сети TCP/IP, том 3. Разработка приложений типа клиент/сервер для Linux/POSIX / Д. Стивенс, Д. Камер – Омск: Диалектика, 2007. – 592 с.

29. Барфилд Э., Уолтерс Б. Программирование клиент-сервер в локальных вычислительных сетях / Э. Барфилд, Б. Уолтерс – М.: Информационно-издательский дом "Филинъ", 1997. - 424 с. Пер. с англ. ISBN 5-89568-015-1

30. Richards D. Linux Thin Client Networks Design and Deployment / D. Richards – Packt Publishing, 2008. - 173 p.

31. Implementing Distributed Systems – Client-Server Technology [Електронний ресурс] / W3Computing– 2011. –Режим доступу до ресурсу https://www.w3computing.com/systemsanalysis/implementing-distributed-systems-client-server-technology/

32. Taylor Ian J., Harrison A. From P2P to Web Services and Grids. Peers in a Client/server World / Ian J. Taylor, A. Harrison – London: Springer-Verlag, 2005. – 269 p.

33. Smith P. From P2P to Web Services and Grids. Peers in a Client/server World / P. Smith – USA: Sams Indianapolis, 1994. – 365 p.

34. Грід обчислення [Електронний ресурс] / Wikipedia – 2017. – Режим доступу до ресурсу https://uk.wikipedia.org/wiki/%D2%90%D1%80%D1%96%D0%B4_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F

35. Thompson J. Future Perspectives: High-Performance Computing // Statistics for Bioinformatics Vol. 10, Issue 4. 2016. P. 33-70.

36. Mehrdad A.Mizani. Chapter 17 - Cloud-Based Computing // Key Advances in Clinical Informatics. 2017. P. 239-255.

37. Virtual organization (grid computing) [Електронний ресурс] / Wikipedia – 2010. – Режим доступу до ресурсу https://www.revolvy.com/page/Virtual-organization-%28grid-computing%29

38. Peer-to-Peer vs. Client-Server Networks [Електронний ресурс] / Techwalla – 2015. – Режим доступу до ресурсу https://www.techwalla.com/articles/peer-to-peer-vs-client-server-networks

39. Lancett P. Client Server Architecture Advantages & Disadvantages [Електронний ресурс] / P. Lancett // Techwalla : [сайт]. – Режим доступа https://www.techwalla.com/articles/client-server-architecture-advantages-disadvantages. – Назва з екрану.

40. Network Management Model: P2P vs. Client-Server [Електронний ресурс] / InforExchange.com – 2019. – Режим доступу до ресурсу https://www.inforexchange.com/network-management-model-peer2peer-client-server/

41. Galán-Jiménez J, Gazo-Cervero A.: OVERVIEW AND CHALLENGES OF OVERLAY NETWORKS: A SURVEY // International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.1, Feb 2011 P. 12-49.

42. Kamel M., Scoglio C., Easton T: Optimal topology design for overlay networks //. Lectures Notes in Computer Science, Springer, Vol. 4479/ 2007, p. 714–725.

43. Graph (discrete mathematics) [Електронний ресурс] / Wikipedia – 2009. – Режим доступу до ресурсу https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)

44. Kumar S., John J. The Upper and Forcing Connected Edge-to-Vertex Detour Number of a Graph // Journal of Advanced Research in Dynamical and Control Systems. 2018. P. 44-50.

45. Hameed S On Co-regular Signed Graphs // Australasian Journal of Combinatorics. 2014. P. 40-50.

46. Padmanava S Introduction to Graph Theory // Discover the world's research journal. 2011. P. 15-23.

47. Ruohonen K. Graph theory / K Ruohonen – USA: TUT, 2013. – 114 p.

48. SKIENA S.S.: The Algorithm Design Manual. / S.S SKIENA – London: Springer–Verlag, 1998 – 255p.

49. Depth-First Search (DFS) and Depth-First Traversal [Електронний ресурс] / Interview Cake – 2013. – Режим доступу до ресурсу https://www.interviewcake.com/concept/java/dfs

50. Depth-First Search [Електронний ресурс] / OpenGenus IQ – 2012. – Режим доступу до ресурсу: https://iq.opengenus.org/depth-first-search/

51. Depth-first search (DFS) – Algorithms and Data Structures [Електронний ресурс] / CrazyforCode – 2016. – Режим доступу до ресурсу: http://www.crazyforcode.com/depth-first-search-dfs-algorithms-data-structures/

52. All You Need To Know About The Breadth First Search Algorithm[Електронний ресурс] / eduteca! – 2019. – Режим доступу до ресурсу: https://www.edureka.co/blog/breadth-first-search-algorithm/

53. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих / А. Бхаргава – Питер: Питер, 2016. – 288 с.

54. An Overview of Monte Carlo Methods [Електронний ресурс] / Medium – 2018. – Режим доступу до ресурсу: https://towardsdatascience.com/an-overview-of-monte-carlo-methods-675384eb1694

55. Three-forcegraph [Електронний ресурс] / vasturiano – 2019. – Режим доступу до ресурсу: https://github.com/vasturiano/three-forcegraph

56. Дубовой В.М., Грищук Т.В., Ковалюк О.О., Никитенко О.Д., Юхимчук М.С Модель кібер-фізичної системи орднорівневого координаційного керування неперервними розподіленими об’єктами. / В. М. Дубовой, Т. В. Грищук, О. О. Ковалюк, О. Д. Никитенко, М. С. Юхимчук – Вісник Вінницького політехнічного інституту. 2006. № 6, с. 112-115.

57. Наумчук Д.О., Шевчук О.І. Розробка автоматизованої системи управління кешбек сервісом / Д.О. Наумчук, О.І. Шевчук, М.С. Юхимчук // Науково-технічна конференція факультету комп'ютерних систем і автоматики – Рік 2019. - режим доступу : https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2018/paper/viewFile/4858/3990

Додатки

Додаток А

(обов’язковий)

ВНТУ

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Розробка програмного забезпечення системи координнаційного управління. Частина 2. Однорівнева система управління

Студент групи 2АКІТ-18м Шевчук О. І.

“___” _____ 2019 р.

Керівник к.т.н., доцент Юхимчук М. С.

“___” _____ 2019 р.

Вінниця 2019

1. Назва та галузь застосування

1.1. Назва – Інформаційна технологія однорівневого координаційного управління.

1.2. Галузь застосування – Комп'ютеризовані системи управління технологічними процесами.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ № 254 від "_02__" ____10____ 20 19 р.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення якості рішень при однорівневому координаційному управлінні.

4. Вихідні дані для проведення розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. 1. Месарович М., Мако Д., Такахара И. Теория иерархических многоуровневых систем / М. Месарович, Д. Мако, И. Такахара – Москва: МИР, 1973. – 344с.
2. 2 Taylor Ian J., Harrison A. From P2P and Grids to Services on the Web: Evolving Distributed / Ian J. Taylor, A. Harrison – London: Springer, 2009. – 302 p.
3. Глушец В. А., Руппель А. А. Проектирование автоматизированных систем: учебное пособие / В. А. Глушец, А. А. Руппель – Омск: Сибади, 2009. – 145 с.

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- Введення параметрів для побудови графа сусідства точок керування
- Побудова графа сусідства точок керування
- Візуалізація в 3d графа сусідства точок керування
- Налаштування параметрів та вибір режиму оптимізації
- Оптимізація методом Монте-Карло за допомогою хвильового алгоритму

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- Windows, Android; Linux, IOS;
- Браузер із ввімкненим javascript.

5.2.2. Умови експлуатації системи:

- робота на стандартних ПЕОМ в приміщеннях зі стандартними умовами;
- можливість цілодобового функціонування системи;
- текст програмного забезпечення системи є цілком закритим.

6. Економічні показники

До економічних показників входять:
- прогнозування витрат не більше 36102,4 грн..;
- загальна величина витрат не менше 401138 грн..;
- коефіцієнт наукової значимості 0.43;
- особистий внесок магістранта. 0,79;

7. Стадії та етапи розробки.

7.1 Пояснювальна записка:

1	Аналіз предметної області та варіантів побудови архітектури систем. Постановка задач дослідження	12.09.2019р.
2	Розробка алгоритмічної частини та UML діаграм	22.09.2019р.
3	Практична реалізація та аналіз отриманих результатів	3.10.2019р.
4	Підготовка економічної частини	12.11.2019р.
5	Апробація результатів дослідження	22.11.2019р.
6	Публікації	
7	Оформлення пояснювальної записки, графічного матеріалу і презентації	30.11.2019р.
8	Захист МКР	12.12.2019р.

7.2 Графічні матеріали:

- алгоритм пошуку для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення : «01 » 12. 2019 р.
- UML діаграма варіантів використання: «01 » 12. 2019 р.
- UML-діаграма послідовності: «03 » 12. 2019 р.
- Відображення результатів оптимізації: «03 » 12. 2019 р.

8. Порядок контролю і приймання.

8.1. Хід виконання магістерської кваліфікаційної роботи контролюється керівником роботи, консультантами з економічної частини. Рубіжний контроль провести до «7» грудня 2019 р.

8.2. Атестація проекту здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «10» грудня 2019 р.

8.3. Підсумкове рішення щодо оцінки якості виконання магістерської кваліфікаційної роботи приймається на засіданні ДЕК. Захист магістерської кваліфікаційної роботи провести «12» грудня 2019 р.

Додаток Б – Лістинги програм

(обов’язковий)

Код головного модуля програми

```
import 'normalize.css/normalize.css'
;
import './styles/index.scss';
import {MDCTextField} from '@materia
l/textfield';
import { callculeteGraphData } from
'./app/index';
import { getObjForGraph } from './ap
p/createGraphObj';
import { genCoord } from './app/genC
oord';
import { getGraphDataSets } from './
myGraph/data-set-loader.js';
import { graphCreator } from './app/
traverseGraph/three';
import { startOptimization } from '.
/app/optimiz/main';
import { createMatrixInDOM } from '.
/app/createMatrix';

//матриця суміжності
let adjancencyMatrix = [];
//матриця із відстанями
let distancesMatrix = [];

//додаємо логіку для кнопки Побудува
ти граф
const genGraphBtn = document.getElem
entById('genGraphBtn');

genGraphBtn.addEventListener('click'
, async() => {
    // очистюємо таблицю із матрицею
 суміжності та координатами
    const adjancencyMatrixDOM = docu
ment.getElementById('adjancencyTable
');
    const distanceMatrixDOM = docume
nt.getElementById('distanceTable');
    adjancencyMatrixDOM.innerHTML =
'';
    distanceMatrixDOM.innerHTML = ''
;
    // отримуємо координати точок ке
рування із таблиці
    const points = [];
    const f0 = []; // масив значенн
 функції fo
    const tableRows = tBody.rows;
    for(let i = 0; i < tableRows.len
gth; i++) {
        points.push([]);
        for(let j = 1; j < tableRows
[i].cells.length - 1; j++) {
            if(tableRows[i].cells[j]
.innerHTML == '') {
                alert('Заповніть усі
 координати точок керування!');
                return;
            }
            points[i].push(+tableRow
s[i].cells[j].innerHTML);
        }
        f0.push(+tableRows[i].cells[
4].innerHTML);
    }
    //отримуємо значення точності із
 інпута
    const precision = precisionInput
GM.input_.value;
    if(!isFinite(precision) || +prec
ision <= 0) return;
    // отримуємо матрицю суміжності
з координат
    const { coordinates, neigMatrix,
 distances } = await callculeteGraph
Data({points, precision, f0});
    adjancencyMatrix = neigMatrix.sl
ice();
    distancesMatrix = distances.slic
e();
    //додаємо дані матриці суміжност
і в таблицю
    createMatrixInDOM(adjancencyMatr
ixDOM, adjancencyMatrix, 'Матриця су
міжності');
    createMatrixInDOM(distanceMatrix
DOM, distancesMatrix, 'Матриця відст
аней');

    const optDiv = document.getEleme
ntsByClassName('optimization-
block')[0];
    optDiv.style.display = 'block';
    const graphHeader = document.get
ElementById('graphHeader');
    graphHeader.style.display = 'blo
ck';
    //підготовлюємо об'єкт для побуд
ови графа передаючи в
    //функцію getObjForGraph(координ
ати, матрицю суміжності)
    const graphData = getObjForGraph
(coordinates, neigMatrix);
    //const graphData = await callcu
leteGraphData({points, precision});
    const Graph = ForceGraph3D()
    (document.getElementById("3d-
graph"));

    let curDataSetIdx;
```

```
    //отримуємо налаштування графа
    const dataSets = getGraphDataSet
s(graphData);

    //функція що перебудовує граф пр
и зміні вхідних данних
    let toggleData;
    (toggleData = function() {
        curDataSetIdx = curDataSetId
x === undefined ? 0 : (curDataSetIdx
+1)%dataSets.length;
        const dataSet = dataSets[cur
DataSetIdx];

        Graph.resetProps(); // Wipe
current state
        dataSet(Graph); // Load data
 set
    })(); // IIFE init
})

// перевірка введенних данних на чис
ло в таблицю
let cordTable = document.getElementB
yId('coordTable');
let oldCellValue = '';
cordTable.addEventListener('keydown'
, (event) => {
    oldCellValue = event.target.inne
rHTML;
    setTimeout(()=> {
        if(!isFinite(+event.target.i
nnerHTML)) event.target.innerHTML =
oldCellValue;
        console.log(isFinite(+event.
target.innerHTML))
    })
});

//заповнення таблиці данними
//після натиснення кнопки згенерувати
и координати табличка автоматично за
повнюється
const tBody = cordTable.getElementsB
yTagName('TBODY')[0];
const genCoordBtn = document.getElem
entById('genCoordBtn');
genCoordBtn.addEventListener('click'
, () => {
    const coordinates = genCoord(nod
eCount);
    tBody.innerHTML = '';
    for(let i = 0; i < coordinates.l
ength; i++) {
        let newRow = tBody.insertRow
(i);
        newRow.classList.add('mdc-
data-table__row');
        var pointNumberCell = newRow
.insertCell(0);
        pointNumberCell.classList.ad
d('mdc-data-table__cell', 'mdc-data-
table__cell--numeric');
        var num = document.createTex
tNode(i + 1);
        pointNumberCell.appendChild(
num)
        for(let j = 0; j < coordinat
es[i].length; j++) {
            var newCell = newRow.ins
ertCell(j+1);
            newCell.classList.add('m
dc-data-table__cell', 'mdc-data-
table__cell--numeric')
                newCell.setAttribute
('contenteditable', 'true');
                var newText = docume
nt.createTextNode(coordinates[i][j])
;
            newCell.appendChild(newT
ext);
        }
    }

})

// налаштування inputs
// input із кількістю вузлів
// після зміни значення змінюється р
озмір таблички із координатами
const nodeCountInput = new MDCTextFi
eld(document.getElementById('qntInpu
t'));
let nodeCount = 0;
nodeCountInput.input_.onchange = (ev
ent) => {
    tBody.innerHTML = '';
    if(parseInt(event.target.value))
 {
        nodeCount = event.target.val
ue;
        for(let i = 0; i < nodeCount
; i++) {
            let newRow = tBody.inser
tRow(i);
            newRow.classList.add('md
c-data-table__row');
            let pointNumberCell = ne
wRow.insertCell(0);
            pointNumberCell.classLis
t.add('mdc-data-table__cell', 'mdc-
data-table__cell--numeric');
            let num = document.creat
eTextNode(i + 1);
            pointNumberCell.appendCh
ild(num);
            for(let j = 0; j < 4; j+
+) {
                var newCell = newRow
.insertCell(j+1);
                newCell.classList.ad
d('mdc-data-table__cell', 'mdc-data-
table__cell--numeric')
                newCell.setAttribute
('contenteditable', 'true');
            }
        }
```

```
    } else {
        nodeCount = 0;
    }
};

// input із точністю

const precisionInputGM = new MDCText
Field(document.getElementById('preci
sionInput'));
precisionInputGM.input_.onblur = (ev
ent) => {
    let input = event.target;
    if(!isFinite(input.value)) {
        precisionInputGM.root_.class
List.add('mdc-text-field--invalid');
    }
}

// Налаштовуємо оптимізацію
//інпут для стартової точки із якої
буде починатися обхід в ширину графа
const startPointInput = new MDCTextF
ield(document.getElementById('startO
ptPointInput'));
let targetOptPoint = 1;
startPointInput.input_.onchange = (e
vent) => {
    const value = event.target.value
;
    if(parseInt(value) || value == 0
) {
        targetOptPoint = +value;
    }
}

const inputsArr = [];
const startBetaInput = new MDCTextFi
eld(document.getElementById('startOp
tBeta0Input'));
inputsArr.push(startBetaInput);
const koeficientKInput = new MDCText
Field(document.getElementById('start
OptKoefKInput'));
inputsArr.push(koeficientKInput);
const KoefBInput = new MDCTextField(
document.getElementById('KoefBInput'
));
inputsArr.push(KoefBInput);
const KoefAInput = new MDCTextField(
document.getElementById('KoefAInput'
));
inputsArr.push(KoefAInput);
const IterationQntInput = new MDCTex
tField(document.getElementById('Iter
ationQntInput'));
inputsArr.push(IterationQntInput);
const desiredPercentageInput = new M
DCTextField(document.getElementById(
'desiredPercentageInput'));
inputsArr.push(desiredPercentageInpu
t);

inputsArr.forEach((element) => {
    console.log(element)
    element.input_.onblur = (event)
=> {
        let input = event.target;
        if(!isFinite(input.value) ||
 +input.value <= 0) {
            element.root_.classList.
add('mdc-text-field--invalid');
        } else {
            element.root_.classList.
remove('mdc-text-field--invalid');
        }
    }
})
const optimizationBtn = document.get
ElementById('startOptimizationBtn');

optimizationBtn.addEventListener('cl
ick', () => {
    if(!adjancencyMatrix.length) ret
urn;

    // перевіряємо інпути із блоку о
птимізації
    let constantsObj = {}; // об'єкт
 із константами із інпутів блоку опт
имізації
    let canBuildGraph = true;
    const inputs = document.querySel
ectorAll('.input_data_js');
    inputs.forEach((element) => {
        const valueName = element.ge
tAttribute('data-name');
        const value = Number(element
.value);
        if (value > 0 && isFinite(va
lue)) {
            element.parentElement.cl
assList.remove('mdc-text-field--
invalid');
            constantsObj = {
                ...constantsObj,
                [valueName]: value,
            };
        } else {
            element.parentElement.cl
assList.add('mdc-text-field--
invalid');
            canBuildGraph = false;
        }
    });
    console.log(constantsObj);

    // хвильовий алгоритм проходженн
я графа,
    // що базується на обході графа
в ширину

    console.log('Порядок проходження
 графа');
    const graph = graphCreator(true)
    if(!adjancencyMatrix.length) ret
urn;
```

```
    for(let i = 0; i < adjancencyMat
rix.length; i++) graph.addNode(i);
    for(let i = 0; i < adjancencyMat
rix.length; i++) {
        for(let j = 0; j < adjancenc
yMatrix[i].length; j++) {
            if(adjancencyMatrix[i][j
] == 1) {
                graph.addEdge(i, j)
            }
        }
    }

    const graphOrder =[];

    graph.breadthFirst(targetOptPoin
t, node => {
        console.log(node.id)
        graphOrder.push(node.id);
    })

        // створюємо об'єкт для опти
мізації типу {points: [], disctance:
 {} }
        // базуючись на тому, які зн
ачення попали в graphOrder
        const adjObj = {};
        for(let i = 0; i < adjancenc
yMatrix.length; i++) {
            if(!graphOrder.includes(
i)) continue;
            adjObj[i] = { points: []
, distance: {} };
            for(let j = 0; j < adjan
cencyMatrix[i].length; j++) {
                if(adjancencyMatrix[
i][j] == 1) {
                    adjObj[i].points
.push(j)
                    adjObj[i].distan
ce[j] = distancesMatrix[i][j];
                }
            }
        }
        console.log(adjObj);
    if (canBuildGraph) startOptimiza
tion(adjObj, constantsObj, graphOrde
r, currentCountMode);
    if (!canBuildGraph) alert('Запов
ніть усі обовязкові поля!');

});

const TOGGLE_CHANGE_MODE_BTNS = docu
ment.querySelectorAll('#toggle_mode_
js input');
const TOGGLE_CHANGE_MODE_LABELS = do
cument.querySelectorAll('#toggle_mod
e_js label');
let currentCountMode = 'withLimit';

const toggleModeHandler = () => {
    TOGGLE_CHANGE_MODE_BTNS.forEach(
(btn) => {
      btn.addEventListener('click',
() => {
        currentCountMode = btn.getAt
tribute('data-state');
        TOGGLE_CHANGE_MODE_LABELS.fo
rEach(element => element.classList.r
emove('active'));
        btn.parentElement.classList.
add('active');
      });
    });
  };
  toggleModeHandler();
```

Реалізація блоку оптимізації та хвильового алгоритму

```
/* eslint-disable no-await-in-
loop */
/* eslint-disable guard-for-in */
import {
  pow, e,
} from 'mathjs';
import {
  kofeicientZ,
} from './constants';
import transferFunction from './tran
sferFunction';
import getRandom from './randomEffec
t';
import MatrixTable from './MatrixTab
le';
import ResultMatrixTable from './Res
ultMatrixTable';

export function startOptimization (g
raphTopsObject, constantsObj, graphO
rder, currentCountMode) {
  const newBetaKeys = {
    newBetaZeroValue: 'newBetaZero',
    oldBetaZeroValue: 'oldBetaZero',
  };

  const modes = {
    withLimit: 'withLimit',
    whileResult: 'whileResult',
  };

  const { withLimit, whileResult } =
 modes;
  const { newBetaZeroValue, oldBetaZ
eroValue } = newBetaKeys;

  const canSearchBestResultLoop = tr
ue;
  const CREATE_MATRIX_BTN = document
.querySelector('#create_matrix_btn_j
s'); // кнопка на створення матриці
  const START_PROGRESS_BTN = documen
t.querySelector('#start_progress_btn
_js');
  let resultsCOrreletionControlArr =
 []; // в режимі пошуку відносно від
остку бажаного відхилення
```

```
  let bestYoutResultObj = null;
  let currentTopPoint; // змінна буд
е змінюватися під час проходження хв
ильового алгоритму і дорівнюватиме п
оточній вершині
  //  graphTopsObject об'єкт що запо
внюється з матриць і використовуєтьс
я для всіх обрахувань
  let addRandomEffect = false;
  let betaFlag = newBetaZeroValue; /
/ newBetaZero or oldBetaZero keys
  let prevResult = 0; // змінна, яка
 зберігає результат Sigma з минулої
ітерації дял

  // обраховує та повертає новий yOu
t
  // що дорівню X конкретної вершини
 перемножиний на Бета цієї ж вершини
  const countGraphTopsY = ({ graphTo
p, x }) => new Promise((resolve) =>
{
    const beta = graphTop.beta || 1;
    resolve(x * beta);
  });

  // підраховуємо X для конкретної в
ершини для подальшого вирахування yO
ut
  // формула X:
  // сума всіх yOut сусідніх вершин
перемножених на Бета сусідніх вершин
 і інші коефіцієнти які загальні для
 всіх
  // формула в рядку де обраховуєтьс
я result
  const countGraphTopsX = ({ graphTo
p }) => new Promise((resolve) => {
    resolve(graphTop.points.reduce((
accumulator, point) => {
      const currentTop = { ...graphT
opsObject[point] };
      let yOut = currentTop.yOut ||
1;
      if(isFinite(yOut)) yOut = 1;
      const distance = graphTop.dist
ance[point];
      const { beta } = currentTop;
      const currentT = distance / ko
feicientZ;
      const result = yOut * (beta /
(constantsObj.kofeicientA * Number(p
ow(currentT, 3 / 2)) * Number(pow(e,
 Number(pow(distance, 2)) / (constan
tsObj.kofeicientB * currentT)))));
      return accumulator += result;
    }, 0));
  });

  // по черзі обраховується спочатку
 X для кожної вершини
  // потім yOut для кожної вершини
  // фкнція повертає Sigma, яка дорі
внює yOut останньої вершини
  const searcForGraphXandYResults =
async () => {
    let sigma = 0;
    // eslint-disable-next-line no-
restricted-syntax
    for (const key in graphTopsObjec
t) {
      const currentGraphTop = { ...g
raphTopsObject[key] };
      const newX = await countGraphT
opsX({ graphTop: currentGraphTop });
      const newYout = await countGra
phTopsY({ graphTop: currentGraphTop,
 x: newX });
      graphTopsObject = {
        ...graphTopsObject,
        [key]: {
          ...currentGraphTop,
          x: newX,
          yOut: newYout,
        },

      };
      sigma = newYout;
    }
    return new Promise(resolve => re
solve(sigma));
  };

    // обраховуємо Бета для конкретн
ої вершини
  // обраховує deltaD різниця попере
днього Бета нульове  помножене на ра
ндосний вплив
  // ( Бета нульове з минулої ітерац
ії яке ми вирішили використовувати (
нове або старе) )
  // ( рандосний вплив - рандомне чи
сло від 0 до 1)
  // якщо меньше 0.33 - вплив = 0.9
  // якщо від 0.33 до 0.66 - вплив =
 1
  // якщо від 0.66 до 1 - влив = 1.1
  // а також V за формулою
  // Time1 * prevV + deltaD * deltaT
(constanta)
  // ----------------------------
----------
  //           Time1 + deltaT
  const countBeta = ({ beta = 0, V,
currentBetaZero }) => new Promise(as
ync (resolve) => {
    const randomEffect = addRandomEf
fect ? getRandom() : 1;
    // const randomEffect = 1;
    const newBetaZero = currentBetaZ
ero * randomEffect;
    let newBeta = beta;
    let transferFunctionResult = V;
    // console.log('beta zero: ', ne
wBetaZero);
    let iterattionCounter = 0;
```

```
    while ((Math.abs(newBeta - newBe
taZero)) / newBetaZero > 0.1 && iter
attionCounter < 20) {
      const deltaD = newBetaZero - n
ewBeta;
      transferFunctionResult = await
 transferFunction({ deltaD, V: trans
ferFunctionResult });
      newBeta = transferFunctionResu
lt * constantsObj.kofeicientK;
      iterattionCounter++;
    }

    resolve({ newBeta, newV: transfe
rFunctionResult, newBetaZero });
  });

  // проходимось по всім вершинам гр
афа
  // обраховуємо новий Бета, бета ну
льове та V
  // заповнюємо цими даними об'єкт з
 інформацією вершин
  const searcForGraphBetasResults =
async () => {
    // eslint-disable-next-line no-
restricted-syntax
    //for (const key in graphTopsObj
ect) {
      const key = currentTopPoint;
      const currentGraphTop = { ...g
raphTopsObject[key] };
      const beta = currentGraphTop.b
eta || 0;
      const V = currentGraphTop.V ||
 1;
      const currentBetaZero = curren
tGraphTop.betaZero ? currentGraphTop
.betaZero[betaFlag] : constantsObj.s
tartBeta;
      const { newBeta, newV, newBeta
Zero } = await countBeta({
        beta, V, currentBetaZero,
      });

      graphTopsObject = {
        ...graphTopsObject,
        [key]: {
          ...currentGraphTop,
          beta: newBeta,
          V: newV,
          betaZero: {
            newBetaZero,
            oldBetaZero: currentBeta
Zero,
          },
        },
      };
    //}
    return new Promise(resolve => re
solve('object updated'));
  };

  const searchForSigmaWhileTrue = as
ync (counter = 0) => {
    counter++;
    let prevInternalResult = 0;
    console.log('2mode')
    console.log(graphTopsObject)
    if (canSearchBestResultLoop && c
ounter < 1000) {
      let currentResult;
      for(let i = 0; i < graphOrder.
length; i++) {
        currentTopPoint = graphOrder
[i];
        console.log(graphTopsObject[
currentTopPoint])
        await searcForGraphBetasResu
lts(); // count Y and Beta на кожній
 вершині
        currentResult = await searcF
orGraphXandYResults();

        addRandomEffect = true;
        if (currentResult > prevInte
rnalResult) {
          betaFlag = newBetaZeroValu
e;
        } else {
          betaFlag = oldBetaZeroValu
e;
        }

        prevInternalResult = current
Result;

      }

      if (!bestYoutResultObj) {
        bestYoutResultObj = {
          graphTopsObject, prevResul
t, betaFlag, counter,
        };
      }

      if (bestYoutResultObj.prevResu
lt < currentResult) {
        bestYoutResultObj = {
          graphTopsObject,
          prevResult: currentResult,
          betaFlag,
          counter,
        };
      }

      console.log(prevResult);
      addRandomEffect = true;
      if (currentResult <= prevResul
t) {
        const correletion = currentR
esult / prevResult;
        // console.log('correletion'
, correletion);
        // console.log(resultsCOrrel
etionControlArr);
```

```
        if (1 - correletion < consta
ntsObj.desiredPercentage) {
          resultsCOrreletionControlA
rr.push(graphTopsObject);
          if (resultsCOrreletionCont
rolArr.length === 3) {
            console.log(counter);

            console.log(graphTopsObj
ect);
            const resultMatrixDOM =
new ResultMatrixTable({
              graphObject: bestYoutR
esultObj.graphTopsObject,
              title: 'Результати',
              sigma: bestYoutResultO
bj.prevResult,
              parentBlock: '#result_
matrix_block',
              betaFlag: bestYoutResu
ltObj.betaFlag,
              counter: bestYoutResul
tObj.counter,
            });
            console.log(prevResult);
            return true;
          }
        } else {
          resultsCOrreletionControlA
rr = [];
        }
        betaFlag = oldBetaZeroValue;
        prevResult = currentResult;
        searchForSigmaWhileTrue(coun
ter);
      } else {
        resultsCOrreletionControlArr
 = [];
        betaFlag = newBetaZeroValue;

        prevResult = currentResult;
        searchForSigmaWhileTrue(coun
ter);
      }

      // searchForSigmaWhileTrue();
    } else {
      console.log(counter);
      console.log(graphTopsObject);
      const resultMatrixDOM = new Re
sultMatrixTable({
        graphObject: bestYoutResultO
bj.graphTopsObject,
        title: 'Результати',
        sigma: bestYoutResultObj.pre
vResult,
        parentBlock: '#result_matrix
_block',
        betaFlag: bestYoutResultObj.
betaFlag,
        counter: bestYoutResultObj.c
ounter,
      });
      console.log(prevResult);
    }
  };

  const searchForSigmaLimitLoop = as
ync (counter = 0) => {
    if (counter < constantsObj.loopC
ounterLimit) {
      ////////////////////
      let currentResult;
      for(let i = 0; i < graphOrder.
length; i++) {
        currentTopPoint = graphOrder
[i];
        console.log(graphTopsObject[
currentTopPoint])
        await searcForGraphBetasResu
lts(); // count Y and Beta на кожній
 вершині
        currentResult = await searcF
orGraphXandYResults();
              addRandomEffect = true
;
      if (currentResult > prevResult
) {
        betaFlag = newBetaZeroValue;
      } else {
        betaFlag = oldBetaZeroValue;
      }

      prevResult = currentResult;
      }
      console.log('нова ітерація')
      console.log(graphTopsObject)
////////////////////
      if (!bestYoutResultObj) {
        bestYoutResultObj = {
          graphTopsObject, prevResul
t, betaFlag, counter,
        };
      }

      if (bestYoutResultObj.prevResu
lt < currentResult) {
        bestYoutResultObj = {
          graphTopsObject,
          prevResult: currentResult,
          betaFlag,
          counter,
        };
      }

      counter++;

      searchForSigmaLimitLoop(counte
r);
    } else {
      console.log(graphTopsObject);
      const resultMatrix = new Resul
tMatrixTable({
        graphObject: bestYoutResultO
bj.graphTopsObject,
        title: 'Результати',
        sigma: bestYoutResultObj.pre
vResult,
```

```
        parentBlock: '#result_matrix
_block',
        betaFlag: bestYoutResultObj.
betaFlag,
        counter: bestYoutResultObj.c
ounter,
      });
      console.log(prevResult);
    }
  };

  const startProcess = async () => {
    bestYoutResultObj = null;
        if (currentCountMode === wit
hLimit) {
          searchForSigmaLimitLoop();
        } else if (currentCountMode
=== whileResult) {
          searchForSigmaWhileTrue();
        }
    //toggleModeHandler();
    // searchForSigmaLimitLoop();
  };

  startProcess();


}

// на кожній ітерації пошуку сігми (
searchForSigmaLoop) ми обраховуємо Б
ета для всіх вершин графа searcForGr
aphBetasResults
// **** searcForGraphBetasResults **
**
// циклом бігаємо по об'єкту і обрах
овуємо для кожної вершини Бета викор
истовуючи функцію countBeta()
// функція countBeta() обраховує del
taD різниця попереднього Бета та Бет
а на даній ітерації
// а також V за формулою
// Time1 * prevV + deltaD * deltaT(c
onstanta)
// ---------------------------------
--------
//           Time1 + deltaT

// ================================
================================
====
// обрахувавши Бета для кожної верши
ни графу запускаємо функцію searcFor
GraphXandYResults
// ****** searcForGraphXandYResults
******
// ця функція пробігається по кожній
 вершині графу
// вираховує X для кожної вершини на
 основі сусідніх вершин використовуюч
и функцію countGraphTopsX
// а отримавши X обраховує У використ
товуючи функцію countGraphTopsY, яка
 приймає новий X, який обрахували в
попередній функції і yOut ( x * yOut
 )
//
```

Додаток В

(Обов’язковий)

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
д.т.н., проф. В.М. Дубовой
_____
« _____» _____2019 р.

ПЕРЕЛІК
ГРАФІЧНИХ МАТЕРІАЛІВ

для захисту магістерської кваліфікаційної роботи
на тему

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ КООРДИННАЦІЙНОГО УПРАВЛІННЯ. ЧАСТИНА 2. ОДНОРІВНЕВА СИСТЕМА УПРАВЛІННЯ

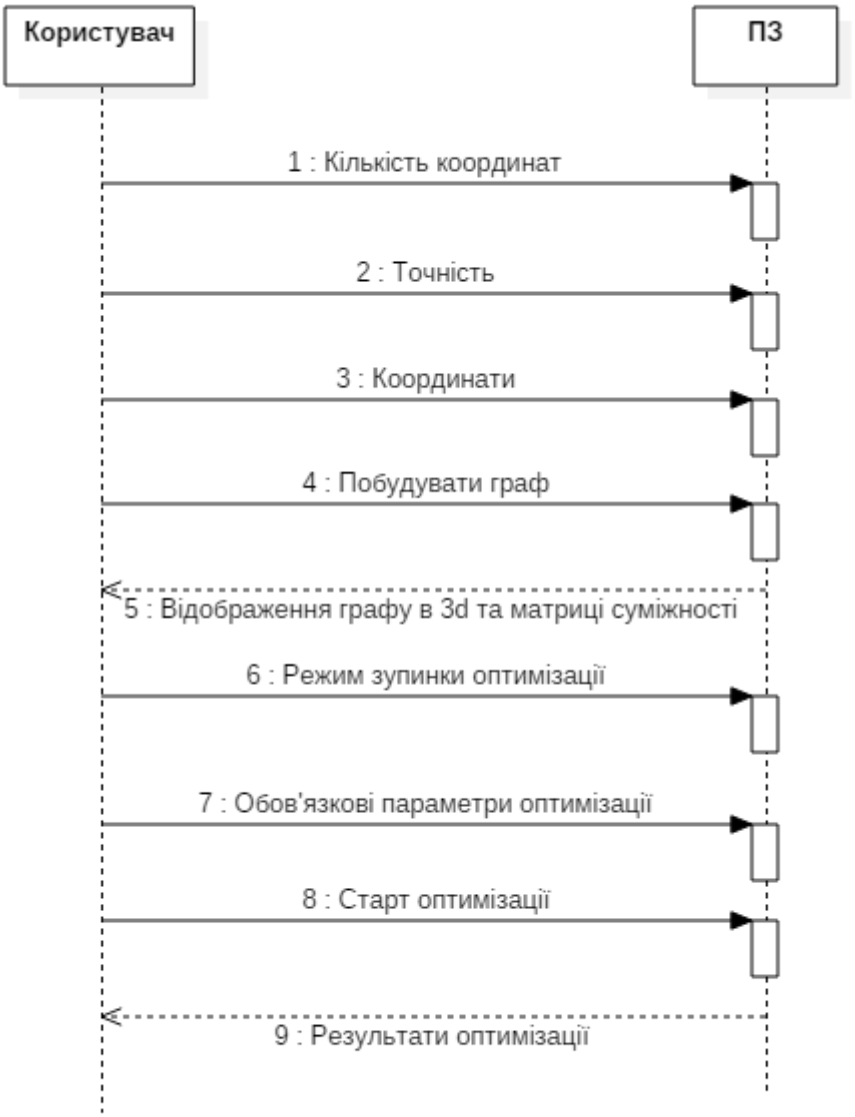
1. Блок-схема алгоритму знаходження для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення.
2. UML-діаграма послідовності.
3. UML-діаграма варіантів використання.
4. Результати оптимізації.

Розробив: Шевчук О. І.
_____ _____
(підпис) (дата)
Перевірив: Юхимчук М. С.
_____ _____
(підпис) (дата)
Рецензент: Маслій Р. В.
_____ _____
(підпис) (дата)

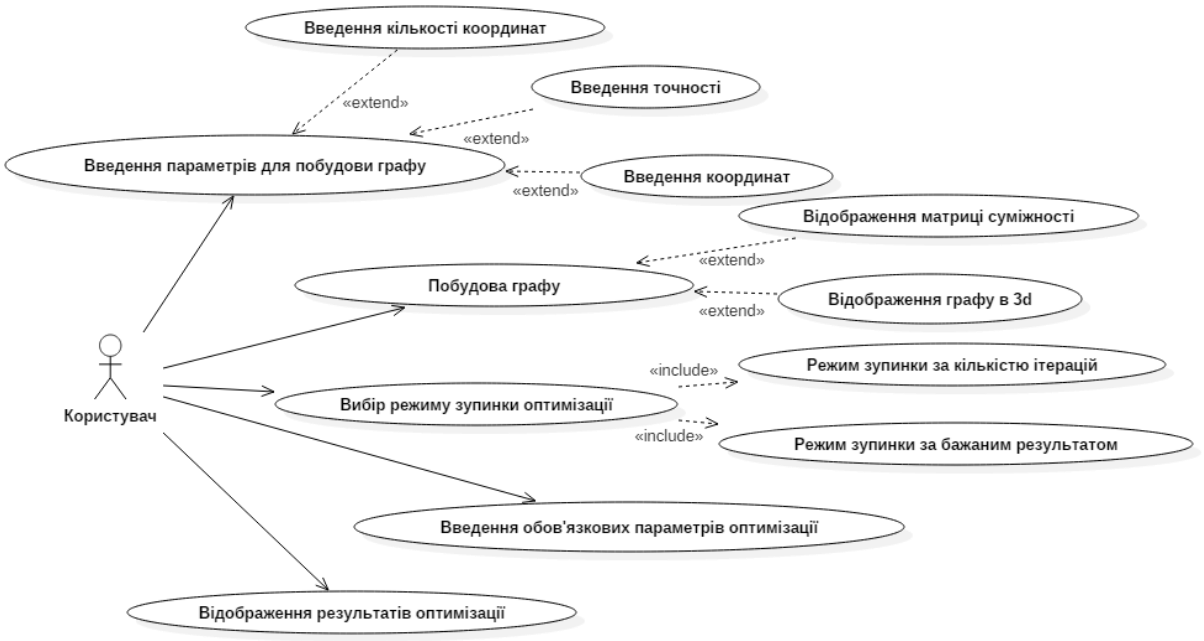
Вінниця 2019

Блок схема алгоритму пошуку для кожної із точок керування однорівневою системою управління сусідніх точок, на які поширюється вплив згідно із заданим законом поширення

UML-діаграма послідовності



UML-діаграма варіантів використання



Результати оптимізації

Моделювання однорівневого координаційного управління

Блок введення параметрів для побудови графу

Введіть кількість точок керування: Кількість 7

Введіть точність ε: Точність 0.1

ПОБУДУВАТИ ГРАФ

Точки керування

Номер точки	X	Y	Z	F_0
1	1	4	0	2
2	3	6	0	6
3	4	4	0	3
4	2	2	0	5
5	6	5	0	9
6	2	5	0	6
7	3	7	0	10

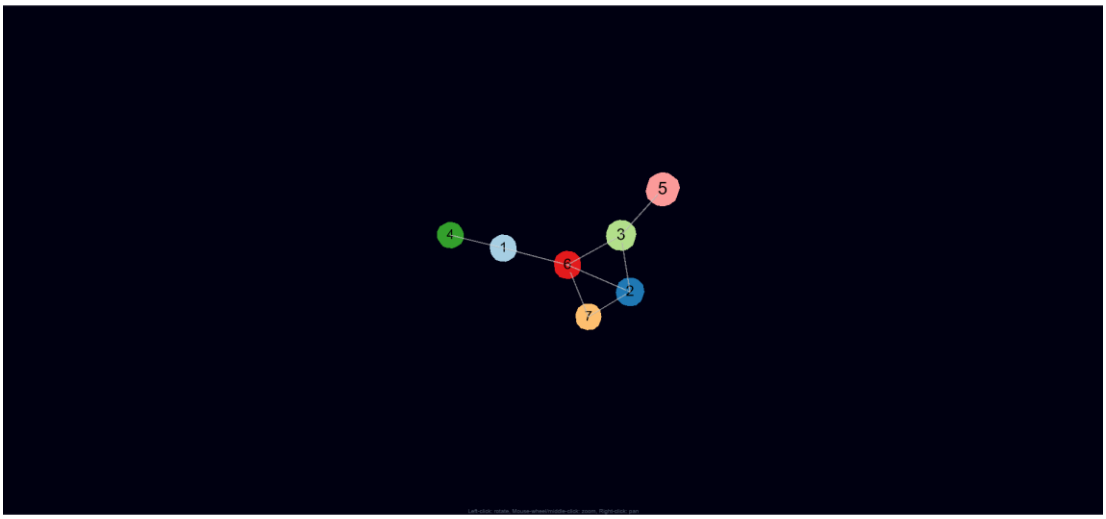
Матриця суміжності

	1	2	3	4	5	6	7
1	0	0	0	1	0	1	0
2	0	0	1	0	0	1	1
3	0	1	0	0	1	1	0
4	1	0	0	0	0	0	0
5	0	0	1	0	0	0	0
6	1	1	1	0	0	0	1
7	0	1	0	0	0	1	0

Матриця відстаней

	1	2	3	4	5	6	7
1	0	2.83	3	2.24	5.1	1.41	3.61
2	2.83	0	2.24	4.12	3.16	1.41	1
3	3	2.24	0	2.83	2.24	2.24	3.16
4	2.24	4.12	2.83	0	5	3	5.1
5	5.1	3.16	2.24	5	0	4	3.61
6	1.41	1.41	2.24	3	4	0	2.24
7	3.61	1	3.16	5.1	3.61	2.24	0

Відображення графу в режимі 3D



Блок введення параметрів оптимізації

Рахувати певну кількість ітерацій | Пошук кращого результату

Введіть номер стартової точки оптимізації: Номер 1

Введіть стартове бета: Beta0 5

Коефіцієнт впливу K: K 4

Коефіцієнт впливу B: B 2

Коефіцієнт впливу A: A 2

Кількість ітерацій: Ітерації 100

Відсоток бажаного відхилення: 10

РОЗПОЧАТИ ОПТИМІЗАЦІЮ

РЕЗУЛЬТАТИ

Максимальний yOut : 21.486082176590823

	Beta	BetaZero	yOut
1	3.5999999999999996	5	3.6162452158549763
2	4.39999999997637	5	24.443587038859583
3	4.511194055073716	5.1263594134884505	2.22537412758606
4	4.000002173834992	5	0.6466749809865078
5	2.584699846693304	2.937157143095721	0.5236305267043421
6	4.000000000011814	5	8.97275843645426
7	8.703130586575089	9.889926007312809	21.486082176590823