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 ієрархічної системи координаційного управління. Частина 1. Ієрархічна система.
08-01.МКР.010.00.000

Студент групи 2АКІТ-18м
Наумчук Д.О.
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

Наумчук Дмитро Олександрович
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Розробка програмного забезпечення ієрархічної системи координаційного управління. Частина 1. Ієрархічна система»

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 ієрархічною системою управління таких таких показників при яких вихідні дані будуть найкращими.

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Розробка методу знаходження впливу для кожної вершини та методу оптимізації., 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. технічних. наук, доцент кафедри ЕПВМ Ратушняк О.Г.		
Розробка алгоритмів та Uml-діаграм програмного забезпечення	доктор технічних наук, професор кафедри КСУ Дубовой В.М.		

Календарний план

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1	Аналаз предметної області та варіантів побудови архітектура систем. Постановка задач дослідження	12.09.2019	
2	Розробка алгоритмічної частини та UML діаграм	22.09.2019	
3	Практична реалізація та аналіз отриманих результатів	3.10.2019	
4	Підготовка економічної частини	12.11.2019	
5	Апробація результатів дослідження	22.11.2019	
6	Публікації		
7	Оформлення пояснювальної записки, графічного матеріалу і презентації	30.11.2019	
8	Захист МКР	12.12.2019	

Дата видачі завдання "_02__" __09___2019 року

Студент _____ _Наумчук Д.О.

Керівник магістерської кваліфікаційної роботи _____ _Юхимчук М. С

АНОТАЦІЯ

Дана магістерська робота присвячена розробці програмного забезпечення ієрархічної системи координаційного управління. Було проведено аналіз предметної області та зроблено варіантний аналіз методів створення архітектури систем. Також було розроблено та реалізовано алгоритм оптимізації за одним критерієм і обмеженнями на параметри використовуючи алгоритм Монте Карло. Розроблено UML діаграми та супровідну документацію. Розроблене програмне забезпечення пройшло тестування, яке підтвердило правильність його роботи.

ANNOTATION

This master's thesis is dedicated to the development of hierarchical coordination management software. A domain analysis was conducted and a variant analysis of methods for creating a system architecture was made. An optimization algorithm for one criterion and parameter constraints was also developed and implemented using the Monte Carlo algorithm. UML diagrams and supporting documentation have been developed. The developed software has been tested to confirm its correct operation.

ЗМІСТ

ВСТУП

Актуальність проблеми. У наш час достатньо прогресивно розвиваються інформаційні і комп’ютерні технології, системний аналіз та інші русла. Це призводить до прогресивного впровадження автоматизації у всі сфери нашого життя. Особливо це відчувається у автоматизації повноцінних масштабних виробництвах. Прикладами вирішення подібних задач виступають як автоматизовані системи управління великою кількістю енергетичних, металургійних чи хімічних виробництв, так і архітектурні рішення побудови цих систем. Коли доходить процес до автоматизації та впровадження новинок саме на таких організаціях в першу чергу звертається увага на 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 координаційного управління.

Тема даної роботи (перша частина комплексної роботи): Частина 1. ієрархічної системи координаційного управління.

Метою магістерської кваліфікаційної роботи є підвищення якості рішень при оптимізації для систем з координаційним управлінням.

Об’єкт дослідження – процес моделювання та оптимізації даних з одним критерієм

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

Іноваційність одержаних результатів.

1. Розроблено алгоритм оптимізації систем з ієрархічним управлінням за одним критерієм
2. Розроблено алгоритм припинення обрахування по бажаному відсотку відхилення на певному відрізку ітерацій для не заданої кількості вершин

Практичне значення одержаних результатів полягає в першу чергу в отриманні програмної системи, що дозволяє обрахувати кращий показник вихідних даних на основі матриці суміжності і коефіцієнтів впливу з певної кількості ітерацій або поки оптимізація не стане максимально стабільна і різниця між показниками мінімальна.

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)[51].

По розробленому програмному забезпеченню було подано заявку на авторські права.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1Опис ієрархічних систем управління

Принцип ієрархічної системи управління координацією характеризується тим, що в кожному окремому підрозділі є керуючий механізм, який самостійно виконує всі функції управління процесами. Всі елементи системи розташовані у порядку зверху вниз і всі рішення з верхніх елементів є обов’язковими для нижніх. Керуючий механізм певного підрозділу у свою чергу також являється підпорядкований своєму керуючому механізму. На цій основі створюється ієрархія для систем управління.

Даний принцип може бути застований, як і у економічних, так і у технологічних системах та процесах. Типовим прикладом ієрархічної системи управління є промислові комплекси типу переробляючих заводів, масштабних теплиць з автоматизованим управління кліматичними умовами.

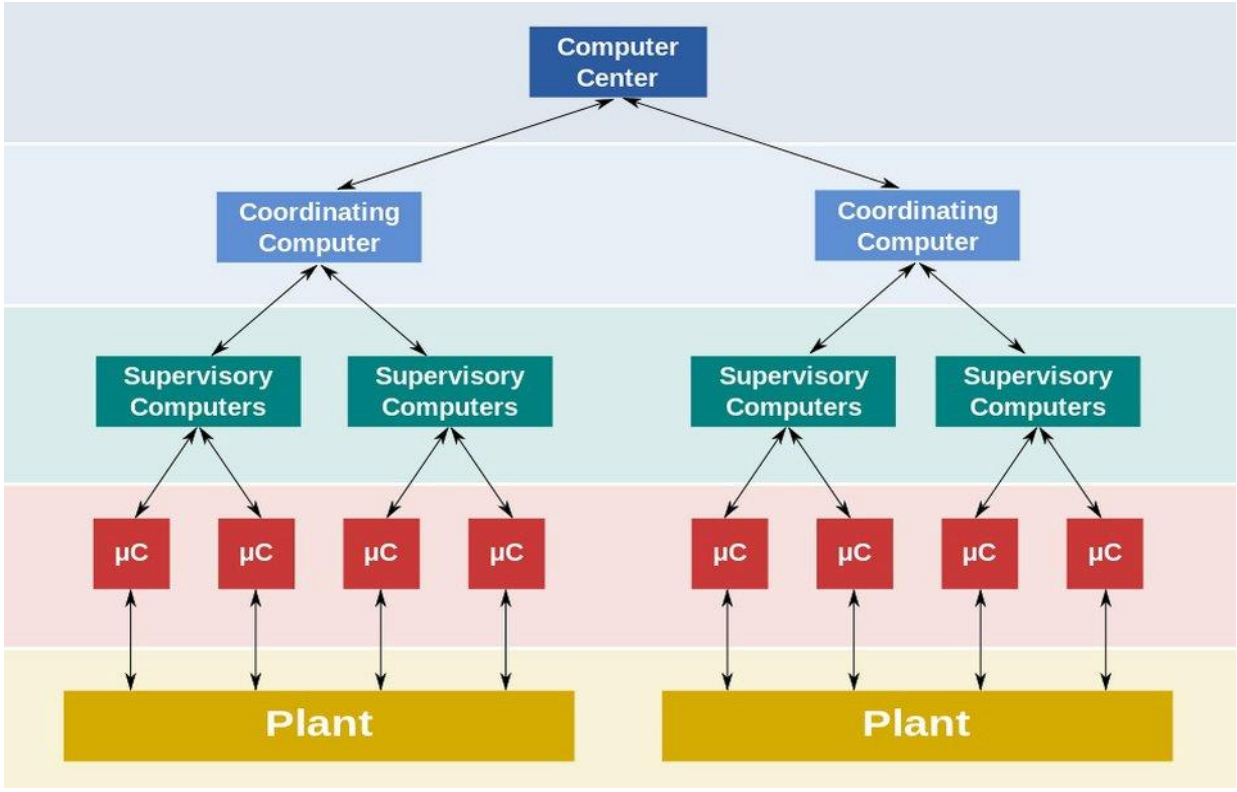


Рисунок 1.1 – Ієрархічна система управління [1]

На рисунку 1.1 схематично представлена типова ієрархічна система управління. Оскільки це схематичне представлення – ми можемо її підставити до будь-якої системи з такою ж системою управління і з мінімальними змінами будемо мати коректний результат. На схемі ми бачимо, як інформація відправляється від першого (найвищого) керівника до наступних, а ті у свою чергу розповсюджують її своїм підлеглим і так до самого нижчого рівня.

Часто ієрархічні структури зустрічаються при вирішенні різних обчислювальних задач, в теорії графів, в математичній логіці, математичній лінгвістиці, евристичному програмуванні та в багатьох інших випадках. Таке широке використання ієрархічні системи управління та їх універсальний характер пояснюється низкою переваг.

Основні переваги ієрархічних систем управління [2]:

1) Свобода для локальних дій
2) Гнучкість систем та можливість легко налаштовувати коректну роботу навіть при змінах

3) Універсальність при вирішенні типових проблем
4) Надійність систем та можливість внесення змін на коремому рівні
5) Немає необхідності пропускати великі потоки інформації через один пункт

Серед недостатків даної системи можна виділити [3]:

1) Велике навантаження на базовий рівень управління
2) Відсутність допоміжних підрозділів
3) Відстуність можливості швидкого вирішення проблем та питаннь між різними структурними підрозділами
4) Рівні контроля обмежені і ця архітектура не включає в себе операцію моніторингу

1.2 Загальний опис систем управління

Так як ми має список переваг та недоліків ієрархічної системи управління можемо зробити аналіз аналогів та обґрунтувати наш вибір системи для вирішення нашої проблеми.

Кожна існуюча система розроблялась для певного ряду задач і обирати архітектуру потрібно детально проаналізувавши задачу. Серед основних пунктів, які ми будем розглядати можна виділити :

1 Вирішення глобальності використання системи. Є задачі які розробляються лише для локального внутрішнього використання, лише для вирішення дуже вузьких задач.

2 Надійність системи. Для навантажених проектів потрібно підбирати максимально оптимізовану архітектуру, адже не завжди зручність розробки виграє у швидкості оброки та передачі даних

3 Гнучкість системи. Можливість легко інтегрувати, масштабувати та змінювати функціонал окремих блоків системи.

4 Модульність. Використання при розробці сучасних модульних технічних засобів.

5 Багаторівневість та обмеженість доступу до них.

Маючи список основних пунктів по яких ми будемо порівнювати системи перейдемо до розгляду аналогів.

1.2.1 Централізована система управління

Першу систему яку ми проаналізуємо – система з централізованим управлінням.

Централізована система управління процесами використовує потужний УВМ (Рисунок 1.2.1) , який управляє декількома технологічними підрозділами.

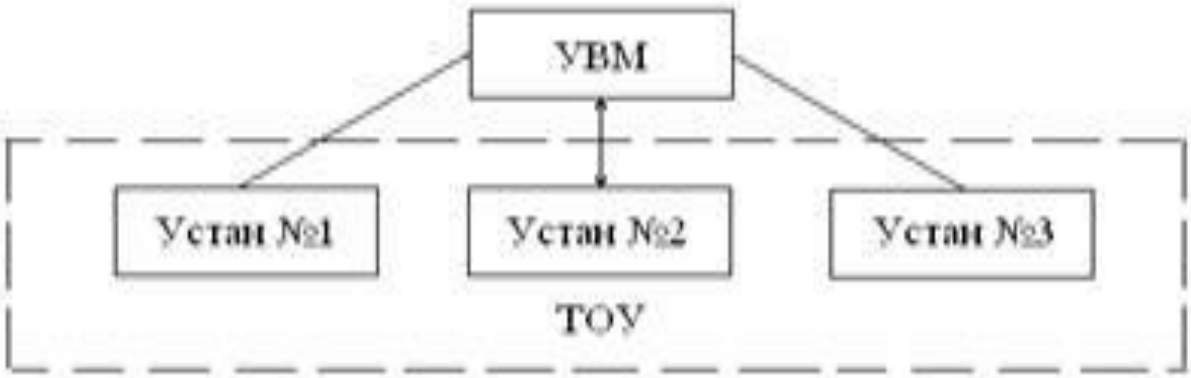


Рисунок 1.2.1 – Централізована система управління [4]

Така конструкція систем управління процесом має суттєві недоліки[6]:

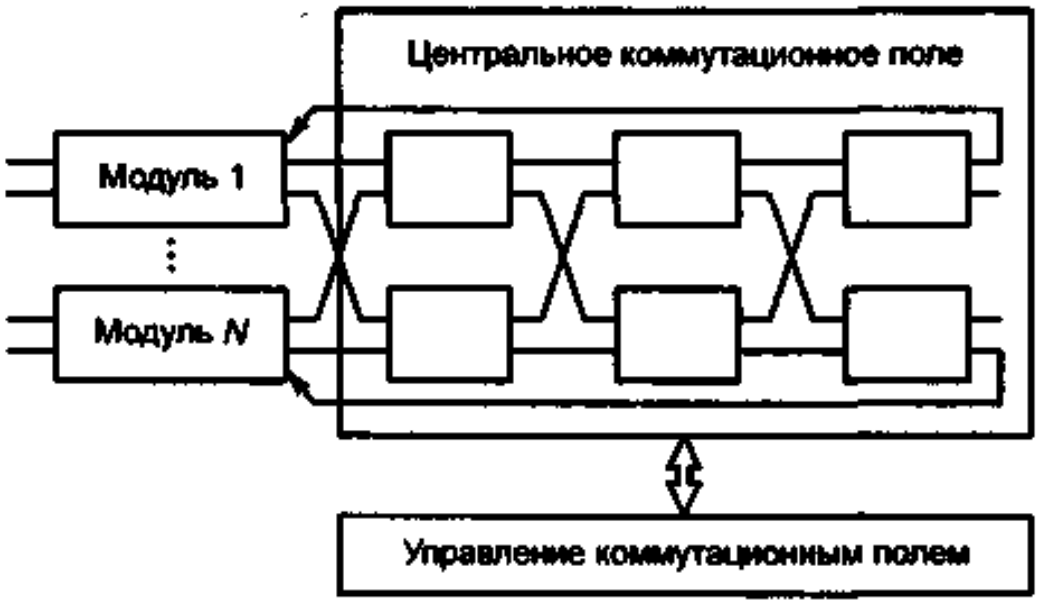
6 Низька надійність - при похибці в УВМ втрачається більшість функцій управління. У цьому випадку виникає необхідність резервного копіювання УВМ.

7 Складність програмного забезпечення.

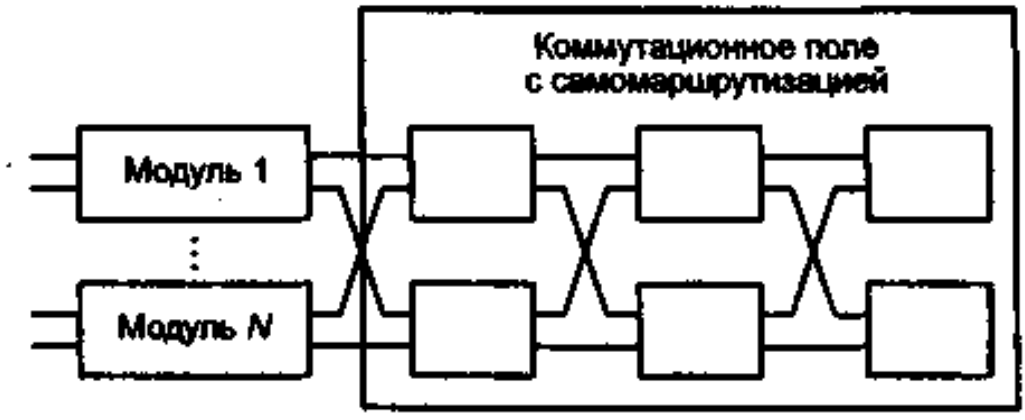
8 Висока вартість зв'язку та обладнання для органічного спілкування комп'ютера з об'єктом управління. Центральне положення УВМ передбачає наявність великої кількості каналів зв'язку, які мають достатній розмір. При такій структурі систем управління процесами вартість інформаційної підсистеми становить до 75% від вартості всієї системи.

1.2.2 Децентралізована система управління

При розгляді децентралізованих систем управління зазвичай підкреслюється територіальний розподіл комп'ютерних систем. Точніше, однак, слід говорити про розподіл функцій. Відстань між окремими підсистемами не відіграє значної ролі; важливо, що вони працюють автономно [5].



а)



б)

Рис. 2.17. Децентрализованное управление:
а) с применением процессора коммутационного поля;
б) без применения процессора коммутационного поля и односторонним коммутационным полем

Рисунок 1.2.2 – Децентралізована система управління [6]

Децентралізовані системи управління (Рисунок 1.2.2) процесами усувають багато з недоліків, оскільки частина ресурсів обробки даних (мікрокомп'ютери) надходить безпосередньо на об'єкти управління. Такі системи управління (децентралізовані) є більш надійними, високим ступенем живучості та меншими витратами.

Перехід до децентралізованої системи управління процесом може бути виправданий використанням нової мікропроцесорної технології (Unicon, Micron[17] тощо), яка має ряд переваг[6]:

- функціональна гнучкість;
- висока швидкість;
- висока надійність;
- легке обслуговування;
- можливість розширення системи шляхом її збільшення (комп'ютерна мережа);
- невеликі розміри та відсутність значного споживання електроенергії.
- відповідальність керівників центрів відповідальності приймати управлінські рішення.

1.3 Аналоги з ієрархічними системами управління

Розглянувши детально ієрархічні системи управління, а також її аналогів перейдемо до реальних прикладів системи з ієрархічною архітектурою управління та проаналізуємо переваги та недоліки вибору даної архітектури.

1.1.1. Металургійний комбінат

Підприємства такого роду широко використовують електронні цифрові обчислювальні машини[16] (далі ЕЦОМ) для керуванням виробництвом.

На Рисунок 1.3.1 зображена схема управління металургійним комбінатом. Комбінат включає коксові печі та майстерні: зарядка, чавун, плавка сталі, плита, гаряча та холодна прокатка та обробка продуктів.

ЕЦОМ верхнього рівня (1) призначений для вирішення загальних проблем планування, фінансів, прогнозування, контролю запасів та інших завдань суто організаційного, а не технологічного характеру. ЕЦОМ третього рівня (2 і 3) використовуються для підготовки довгострокових календарних планів заводу, один з них (2) призначений для планування роботи підготовчих майстерень (коксу, чавуну і сталі), а за допомогою іншого - (3) робочі плани для решти магазинів та секцій [8].

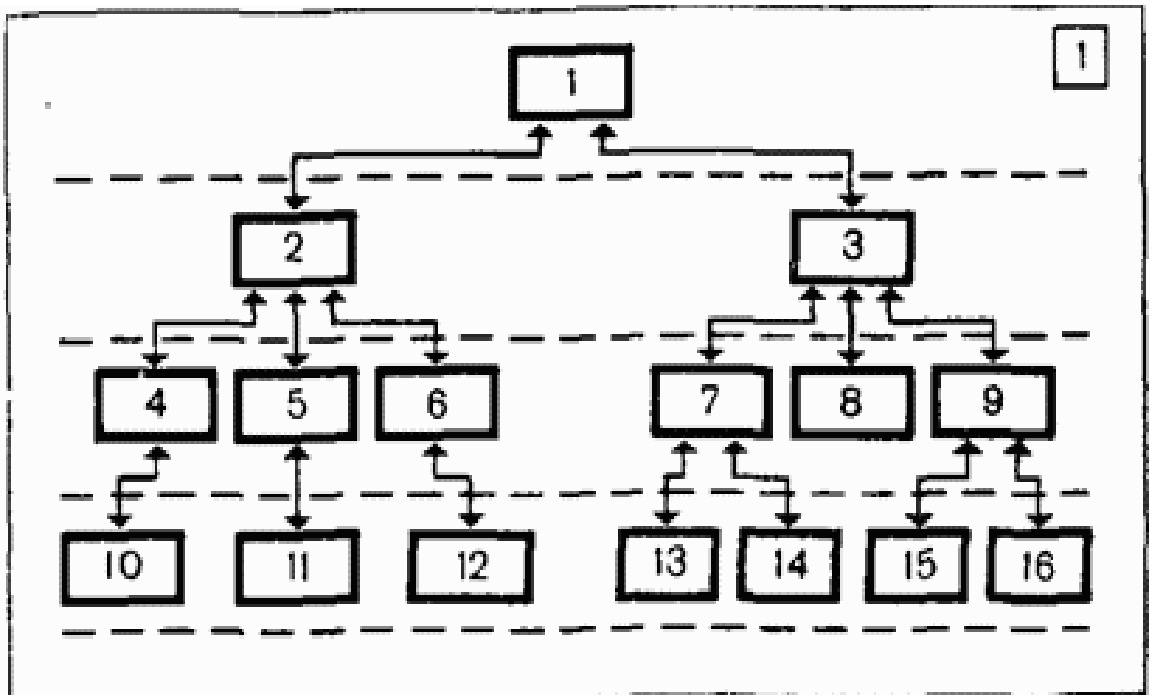


Рисунок 1.3.1 – Ієрархічна система управління металургійним комбінатом[7]

За допомогою ЕЦОМ другого рівня (4-9) для кожного семінару розробляються короткострокові детальні плани роботи, а інформація збирається та обробляється у міру необхідності для процесу автоматичного управління майстернями та їх окремими секціями. ЕЦОМ 1-го рівня (10-16) призначені для автоматичного управління технологічними процесами та окремими пристроями (зарядні машини, доменні печі, перетворювачі тощо).

1.1.2. Роботизований технічний процес для обробки на ньому деталей типу фланець

Вся система управління розбивається на три рівні.

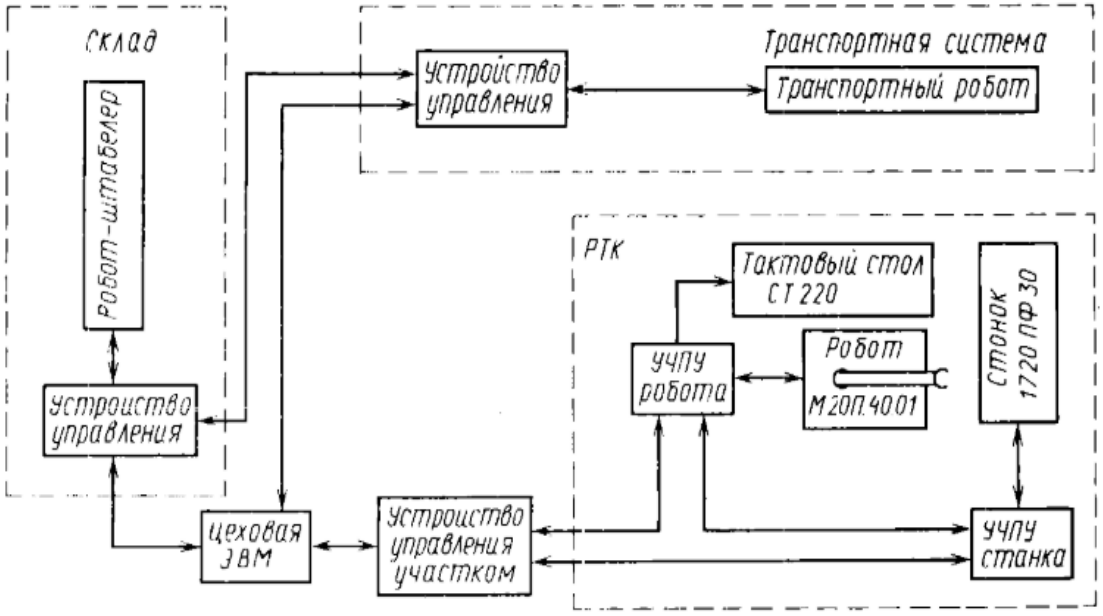


Рисунок 1.3.2 – Ієрархічна система управління процесом для обробки на ньому деталей типу фланець[9]

Перший рівень управління. для управління кількома спільно функціонують об'єктами, в даному випадку управління осередком ГІС, РТК, який об'єднує робот, верстат, тактовий стіл, накопичувач інструментів, контрольно-вимірювальні пристрої, використовують, як правило ЕОМ[15], які координують роботу об'єктів шляхом передачі керуючої інформації контролерам і мікро ЕОМ. Координацію роботи осередків технологічної лінії, транспортних засобів та інших однотипних об'єктів можна розглядати як функції управління другого рівня. На цьому рівні здійснюється контроль і діагностика засобів управління нижнього рівня і відповідного обладнання для проведе-ня профілактичних, налагоджувальних і ремонтних робіт [10].

Управління на другому рівні забезпечує виконання не тільки робочих функцій, але і ряду додаткових, для експлуатації яких використовують, як

правило, ЕОМ, з зовнішньою пам'яттю для розміщення програм функціонування, контролю і діагностики технічних засобів і засобами сполучення з апаратно - настроювачами обладнання, з дисплеєм.

Координація роботи всіх підсистем ГІС є функцією третього рівня управління. на якому зазвичай використовують з розвиненою зовнішньою пам'яттю, необхідної для зберігання програми ЕОМ всіх трьох рівнів управління і даних, їх аспекти функціонування ДПС [14]. Вони забезпечує роботу операторів ГПС, ЕОМ і системою управління в цілому та її можливість отримання друкованих звітів.

1.1.3. Структура управління великою універсальною базою

Розглянемо багаторівневу організаційну структуру управління великою універсальною базою (Рисунок 1.3.3). Ця структура відноситься до багатоцільовий системі. На наступному (першому) рівні вирішальним елементом є завідувач базою (складом), на другому рівні функції вирішального елементу виконую диспетчер бази (складу). На даного керівника покладаються основні функції управління протягом зміни. На наступному рівні вирішальні елементи - завідувачі секціями, комірники. Вони виконують функції управління по відношенню до операторів підйомно-транспортних машин, бригадам робітників - вантажників, сортувальників, пакувальників і т. д. Які розташовуються на нижчого рівня даної організаційної структури. На схемі показані також горизонтальні зв'язки між секціями складу, які характеризують їх технологічну взаємодію і напрямки виробничого процесу при прийомі, сортування, транспортування, зберігання, комплектації і видачі вантажу[12].

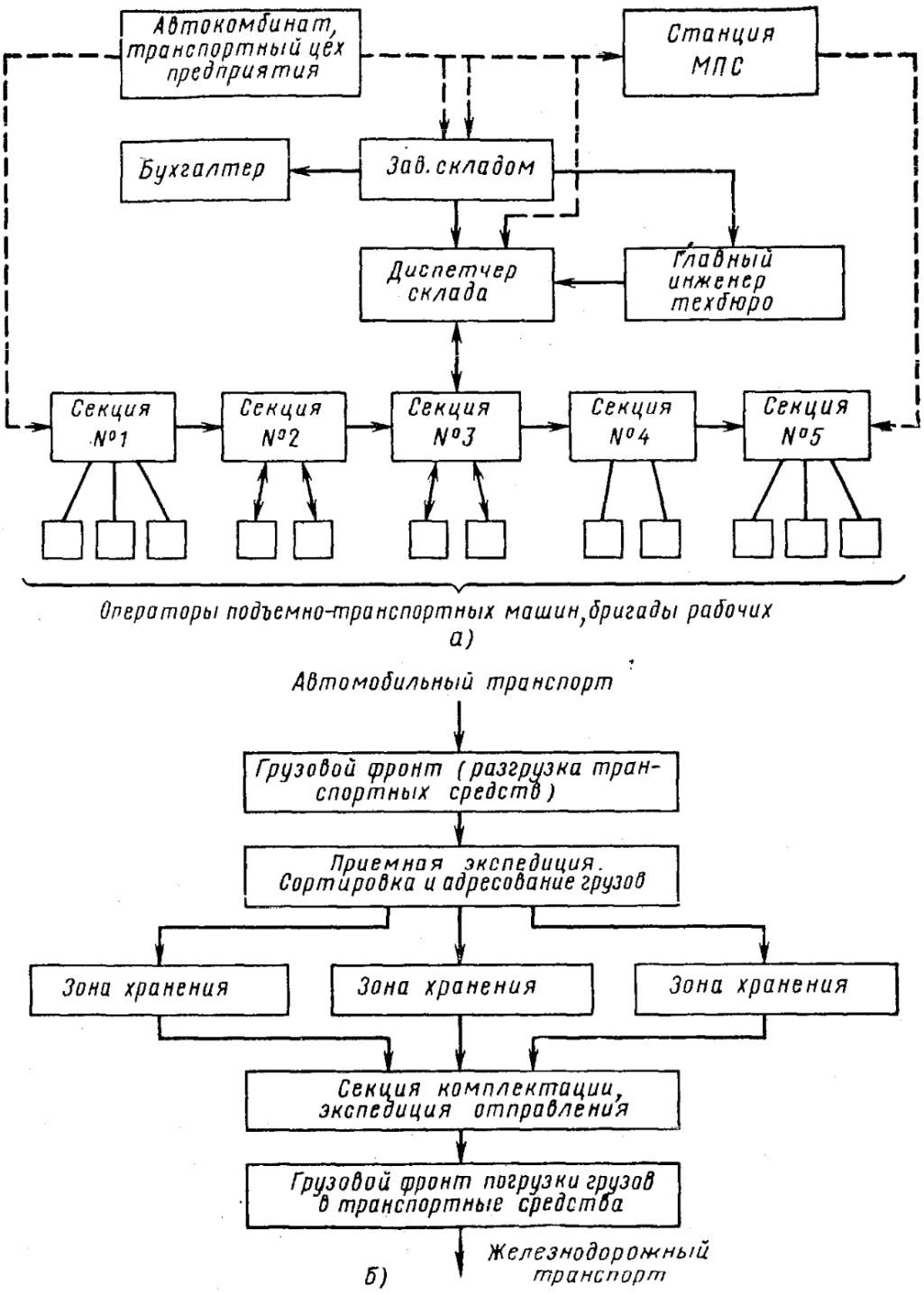


Рис. 4.1. Иерархические системы управления:
а — организационная структура; *б* — функционально-технологическая структура

Рисунок 1.3.3 – Ієрархічна система управління великою універсальною базою [11]

2. РОЗРОБКА АЛГОРИТМІВ ТА UML-ДІАГРАМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Розробка методу знаходження впливу для кожної вершини та методу оптимізації впливів між вершинами для ієрархічних систем координаційного впливу

Для коректної, швидкої та якісної роботи системи потрібно вірно підібрати алгоритм оптимізації. Але перед цим потрібно вистроїти оптимальну, але при цьому якісну логіку побудови графу, обходу всіх вершин, а також зберігання даних.

Для пошуку кращого результату потрібно прогнати якусь певну кількість ітерацій. Було запропоновано два варіанти припинення проходження по графу для отримання і аналізу даних з подальшою оптимізацією, а також зберігання найкращого результату.

2.1.1 Припинення обрахування по бажаному відхиленню

Одним з двох можливих режимів роботи оптимізації являється «Пошук по відсотку відхилення».

В даному режимі всі обрахування відбуваються по тим самим алгоритмам, але особливість цього режиму полягає у тому, що на кожній ітерації ми аналізуємо відсоток відхилення результату отриманого на даній ітерації від попереднього. Якщо відсоток відхилення попадає в межі бажаного відсотку відхилення, тобто являється меншим – ми зберігаємо в спеціальний масив контроля цих змін дані з цієї ітерації. Далі ми дивимось на довжину давного масиву, якщо він більше 3 – дані останні 3 ітерацій змінюються в межах заданого відхилення, а це означає, що ми можемо припиняти роботу оптимізації і виводити дані ітерації з найкращим у output.

У протилежному випадку масив даних очиститься і продовжить роботу алгоритму оптимізації.

Узагальнений алгоритм роботи даного режиму відображено на рисунку 2.1.

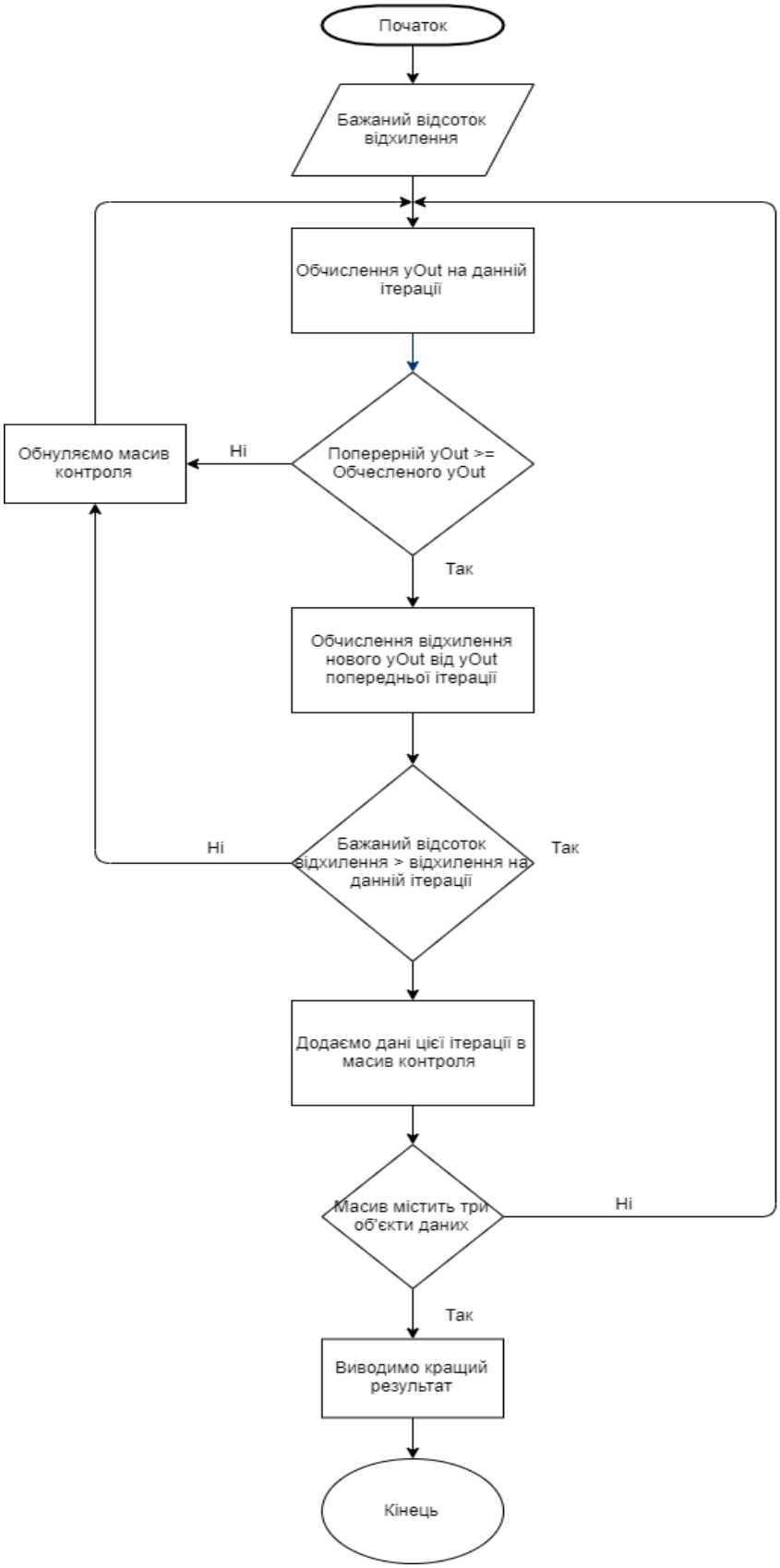


Рисунок 2.1 – Блок-схема алгоритму роботи режиму «Пошук по відсотку відхилення».

2.1.2 Припинення обрахування по заданій кількості проходжень

Даний режим є більш прости у розумінні та реалізації. Оптимізації проходить задану користувачем кількість разів і на останній ітерації видає в інтерфейс користувача дані при яких результат обрахувань у Out був найкращим. Для уникнення помилок є кількість ітерацій по замовчуванню.

Векторна оптимізація, або багатоцільова, представляє напрям в теорії прийняття рішень і вивчає рішення щодо багатьох об'єктивних функцій при плануванні (проектуванні). На відміну від формально однакових проблем з рішеннями в інших областях застосування (ігри проти природи, арбітраж та інші завдання), всі об'єктивні функції визначені, не обрані випадково, але зазвичай не такі важливі [18].

На практиці, крім об’єктивних особливостей, які характеризують якість планів з різних, зазвичай конфліктних, сторін, існує уявлення про глобальну якість різних планів. Однак ця ідея змінюється від завдання до завдання, і важко виділити деякі їх класи, де це було б уніфіковано та формалізовано. У той же час, хоча ідея загального оптимуму іноді дуже розпливчаста, за рахунок додаткових витрат (або ризиків) можна порівняти плани та вирішити, який з них більш прийнятний для особи, яка приймає рішення [19].

Основна інформація в задачі оптимізації вектора - це набір альтернативних X та об'єктивних функцій

2.2 Використання графу для обрахування поширення впливу керування між точками керування

Дана система побудована на принципах роботи з графами, їх логікі проходження, зв’язків та відстаней між вершинами графу.

2.2.1 Визначення графу

Граф — це сукупність об'єктів із зв'язками між ними.

Задати певний граф - означає описати набори з кутами та ребрами, а також їх співвідношення [25].

2.2.2 Методи роботи з графами

В данній задачі було розроблено не зовсім стандартний варіант роботи з графами. Користувач вводить зв’язки заповнюючи стандарту матрицю суміжності[26]. Але у самій програмі ми переводим матрицю в асоціативний масив[27] (далі об’єкт), де ключі являють собі вершини графу. Цей метод більш зручний тим, що ми можемо збігати всі дані про вершину по ключу і перебираючи всі ключі об’єкту – ми одразу отримуємо всю потрібну нам інформацію. Зв’язки між вершинами ми зберігаємо як масив ключі і знову ж таки, коли ми працюємо з певною вершиною графу можемо завжди швидко та зручно отримати потрібні нам дані по ключу.

Коли питання дійшло до варіантів перебору об’єкту – було запропоновано два варіанти :

1) Стандартні методи роботи з об’єктами мови програмування.

У вибраній нами мові програмування є декілька стандартний методів переборів об’єктів, по ключам і просто по значенням

2) Зробити масив ключів, перебирати цей масив і по ключам отримувати дані з об’єкті про кожну з вершин. Цей варіант є більш зручним у випадку, коли на потрібний певний порядок перебору об’єкту або можливі варіанти зміни порядку, такі як реверс перемішання і інші.

Було обрано перший метод, так як наш об’єкт будується по масиву де вершини нумеровані і порядок задається з самого початку від першої до останньої.

2.3 Розробка алгоритму оптимізації, що базується на оптимізації за одним критерієм і обмеженням на параметри

2.3.1 Опис Генетичного алгоритму

Як можливий варіант обрахування кращих показників оптимізації було розглянуто декілька можливих алгоритмів. Один із таких – генетичний алгоритм.

Генетичний алгоритм [20]- це метод вирішення як обмежених, так і необмежених задач оптимізації, який ґрунтується на природному відборі, процес, що рухає біологічною еволюцією.

У генетичному алгоритмі сукупність кандидатних рішень (які називаються індивідами, істотами або фенотипами) розробляється в проблему оптимізації проти кращих рішень. Кожен кандидатський розчин має набір властивостей (його хромосоми чи генотип), які можна мутувати та змінювати; Традиційно рішення представлені у двійковій формі як рядки 0 і 1, але можливі й інші кодування [22].

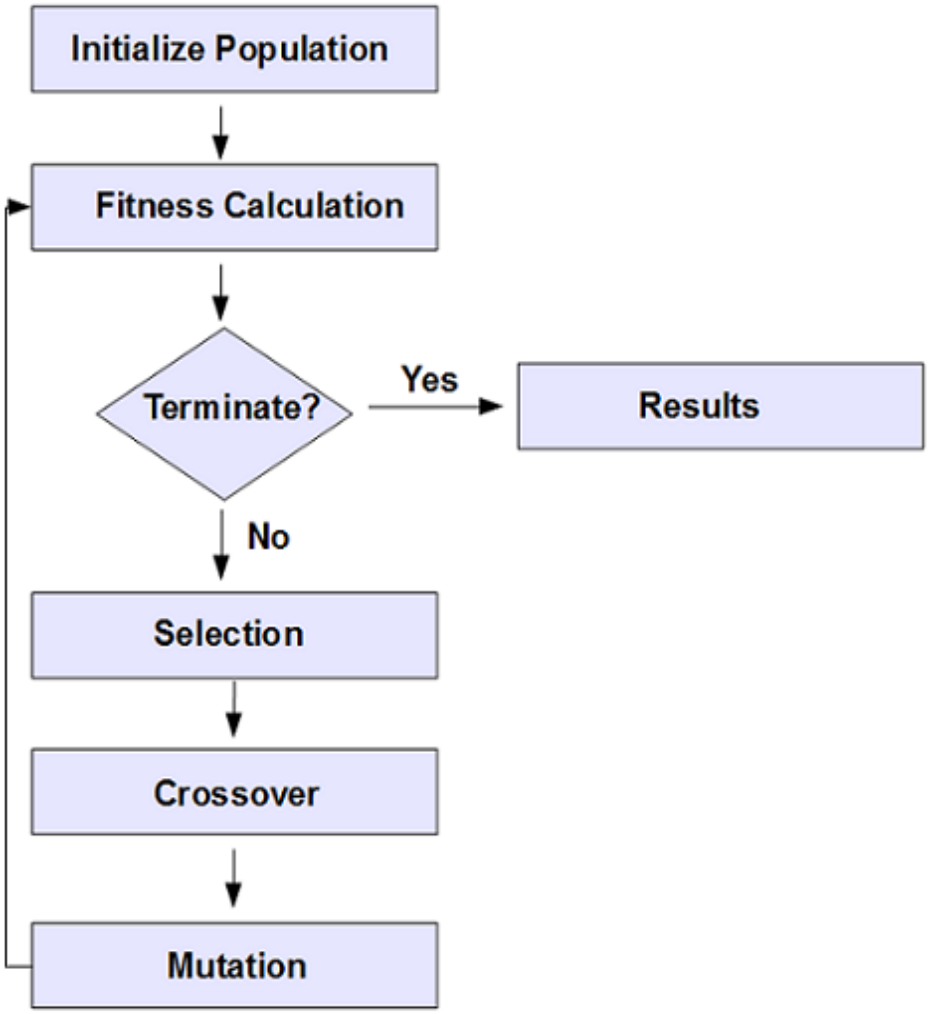


Рисунок 2.2 – Генетичний алгоритм

Генетичний алгоритм неодноразово модифікує сукупність індивідуальних рішень. На кожному кроці генетичний алгоритм відбирає випадково людей із поточної популяції для батьків та використовує їх для отримання дітей для наступного покоління. Протягом наступних поколінь населення "еволюціонує" до оптимального рішення. Ви можете застосувати генетичний алгоритм для вирішення різноманітних проблем оптимізації, які не підходять для стандартних алгоритмів оптимізації, включаючи проблеми, в яких цільова функція є розривною, недиференційованою, стохастичною або дуже нелінійною. Генетичний алгоритм може вирішувати проблеми змішаного цілочисленного

програмування, де деякі компоненти можуть бути цілочисельними. Генетичний алгоритм використовує три основні типи правил на кожному кроці для створення наступного покоління з поточної сукупності [21]:

- Правила відбору вибирають осіб, які називають батьками, які сприяють популяції наступного покоління.
- Правила кросовера поєднують двох батьків для формування дітей для наступного покоління.
- Правила мутації застосовують випадкові зміни до окремих батьків для формування дітей.

2.3.2 Оптимізація методом Монте-Карло

Маючи ієрархічну систему координаційного управління, що складається із n точок управління потрібно розрахувати найкраще у Out на кінцевій точці управління за допомогою оптимізації методом Монте-Карло, де

$$y_{out} = \beta \cdot x \tag{2.1}$$

$$x = \sum_{k=1}^{n} \left\{ y_{out\,k} \frac{\beta_k}{a \cdot t_k^{3/2}} e^{-\frac{r_k^2}{b \cdot t_k}} \right\} \tag{2.2}$$

y_{out} - вихід вершини, яка розглядається;

y_{out} $y_{out\,k}$ - 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 2.3, де

$$d = \beta_0 - \beta \quad (2.3)$$

$$V = \frac{T_1 * V_{\text{попереднє}} + d * \Delta t}{T_1 + \Delta t} \quad (2.4)$$

$$\beta = V * k \quad (2.5)$$

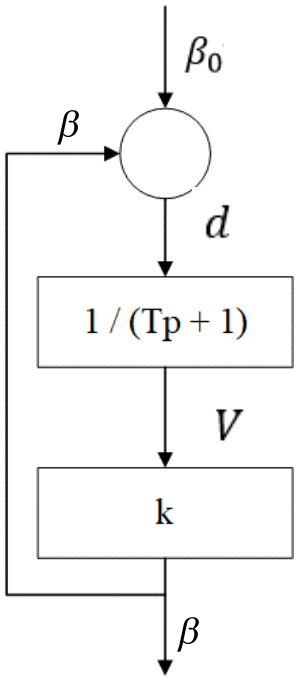


Рисунок 2.3 – Графічне відображення стабілізуючої системи

Монте-Карло - метод чисельного рішення, Математичних задач шляхом моделювання випадкових величин.

Вони дозволяють моделювати складні ситуації, що включають багато випадкових змінних, та оцінювати ефект ризику. Основі галузі використання МК – фізика, теорія ігр та економіка, що призвело у свою чергу до великої кількості відкриттів [49]. Не дивлячись на те, що різноманітність методів Монте-Карло дуже велика - їх об’єднує одна спільна ідея, що всі вони вирішують проблему генерації випадкових чисел для вирішення детермінованих задач [24].

Суть оптимізаційного процесу полягає в підборі таких вхідних β, при яких у Out буде найкращим. Для кожної вершини графу береться нове $\beta 0$ яке у свою чергу дорівнює старому $\beta 0$ помножене на вплив генератора випадковиї чисел. В залежності від згенерованого числа і межей в які він

попадає – ми збільшуємо число на 10%, залишаємо таке саме або збільшуємо на 10%.

Так як розробляється ієрархічна система координаційного управління ми обраховуємо β і нове β_0, що залежить від випадкового впливу для кожної вершини по черзі, щоб потім використати його для обрахування у Out на кожній ітерації.

Отже, ми маємо таку послідовність дій для алгоритму оптимізації показників ієрархічної системи координаційного управління:

1) Заповнюємо матрицю суміжності
2) Заповнюємо матрицю відстаней між вершинами
3) Заповнюємо поля з коефієнтами впливу на вершини при обрахуванні yOut
4) Формуємо об’єкт для проходження і обрахування показників для кожної вершини графу
5) По черзі для кожної вершини в порядку знайденої послідовності проходження вершин графа: генеруємо випадковий вплив на β, обраховуємо β за формулою (2.5), X за формулою (2.2), обраховуємо $У_{out}$ за формулою (2.1) 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;
6) 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;
7) Після закінчення всіх ітерацій виводимо користувачу $У_{out}$ та β найкращого результату

2.4 Розподілені системи як складні об'єкти технічного управління

Розподілені системи як складні об'єкти технічного управління можна оцінювати по-різному. Це пов’язано з багатовимірною взаємодією елементів у системі як між самими елементами, так і з інструментами системи управління. Ці аспекти можна розділити на категорії [41]:

- функціональна взаємодія;

- обмін інформацією;

- енергетична взаємодія.

Розподілене адміністрування системи через процеси прийняття рішень може бути зображене як взаємодія між двома шарами [42].

$$\begin{cases} \{f_i\} : A_i X_i = Y_i \\ \{M_i\} : \bar{y}_i = M_i(x_i) \\ \{F_k\} : d_k = F_k\left(\bar{y}_k, \bar{d}_k, \tau\right) \\ \{E_i\} : \bar{u}_i = E_i(\bar{d}_i) \\ X_i = \bar{u}_i \cup \{x_{ji}, j = 1...n\} \cup \bar{x}_{0i} \\ i = 1...n, \quad k = 1...m \end{cases}$$

Де: n – кількість функціональних елементів;

m – кількість підсистем прийняття рішень;

τ- попередні моменти функціонування системи.

$\{f_i\}$ - множина виробничих функцій;

$\{M_i\}$- функція спостереження;

$\{F_k\}$- функція координації;

$\{E_i\}$- функція управління;

X_i- множина входів;

Y_i- множина виходів.

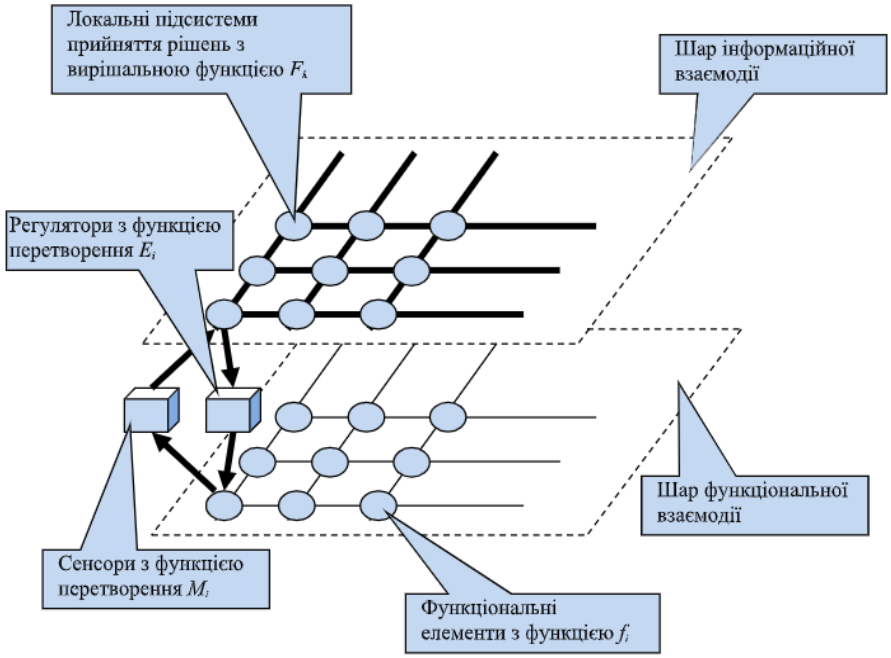


Рисунок 2.4 – Схема взаємодії системи управління з керованою розподіленою системою

2.5 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 кількість вершин графу– користувач вводить кількість бажану кількість вершин графу.

2. Вводити дані для графу – користувач заповняє матрицю суміжності і матрицю відстаней між вершинами графу.

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 таблиця з результатами в вершинах на ітерації з кращим показником вихідних даних.

Розроблена UML діаграма варіантів використання зображена на рисунку 2.5.

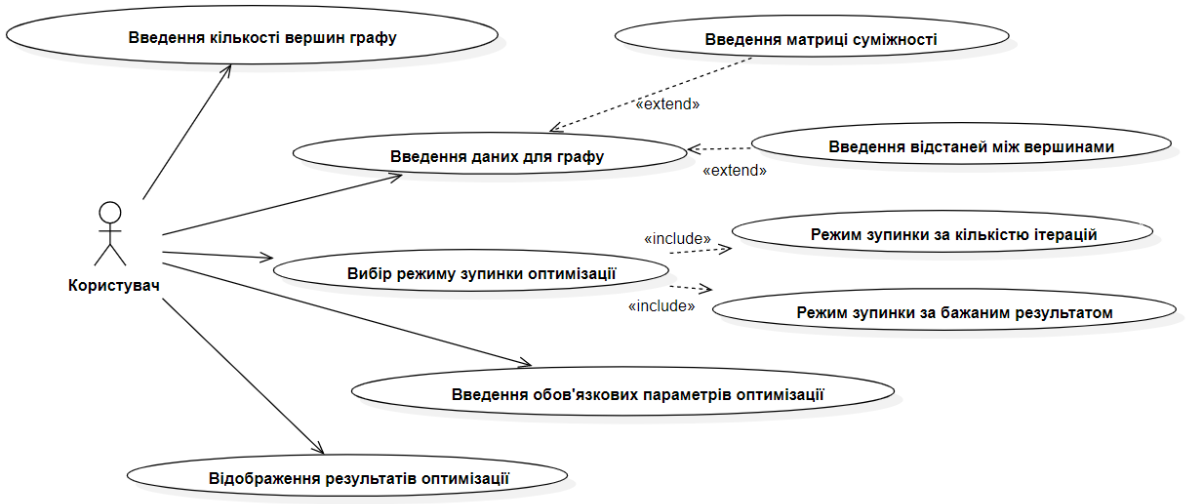


Рисунок 2.5 – UML діаграма варіантів використання

2.4.2. Розробка діаграми 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.

З діаграми видно, що користувач починає взаємодію з поля вводу кількості вершин графу. І якщо введена кількість вершин задовольняє мінімальне потрібне значення програма будує таблиці для вводу даних матриці суміжності і матриці відстаней між вершинами, а також поля вводу коефіцієнтів впливу на похибку. Потім користувач запускає роботу алгоритму і отримує результат всіх обрахувань у таблицю.

Розроблена UML діаграма послідовності зображена на рисунку 2.6.

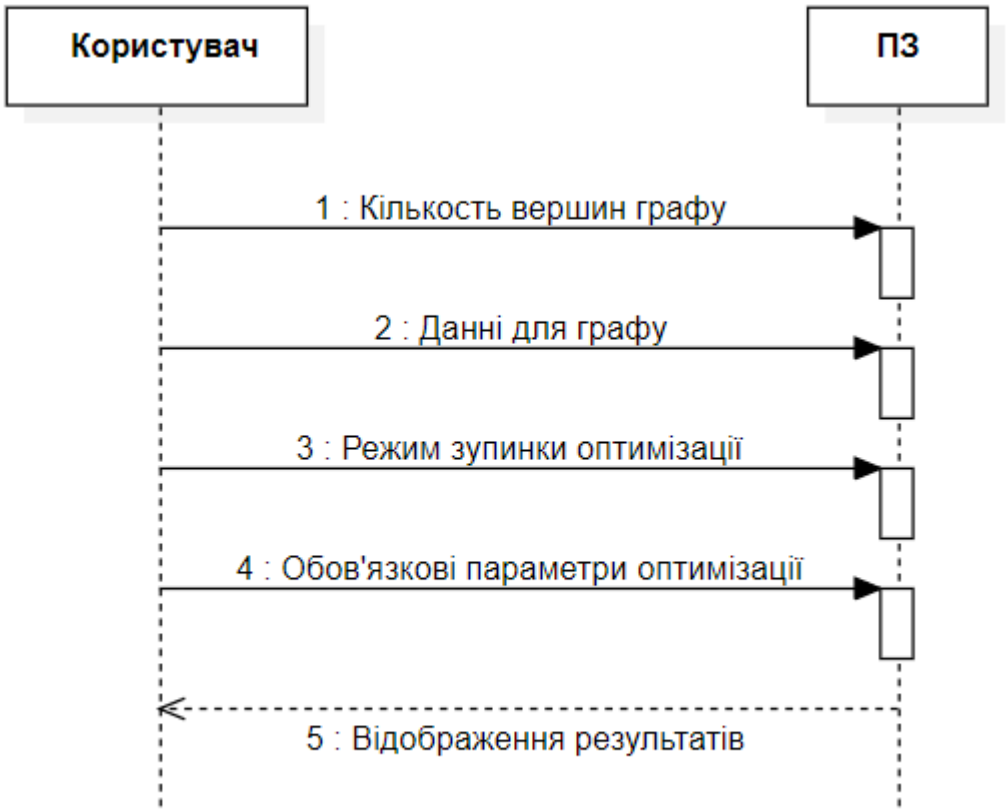


Рисунок 2.5 – UML діаграма послідовності

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Мова програмування

Весь функціонал та обрахунки реалізовані методами мови програмування JavaScript [30]. Особливість цієї мови полягає у тому, що вона підтримує різні парадігми програмування, а саме об’єктно-орієнтовані, імперативні та функціональні стилі. Для розмітки використовується HTML і CSS в парі з Bootstrap (для адаптації на усіх девайсаї).

Основні переваги такого стеку:

1) Мова підходить для виконання браузерних сценарії. Це надає кроссплатформеність
2) Широкий вибір інструментів для тестування та дебагінгу коду функціоналом сучасних браузерів[32]
3) Велика комюніті розробник, що допомагає швидко вирішити питання.
4) HTML і CSS в парі з Bootstrap надає можливість швидко та якісно зробити user friendly інтерфейс [31]

Недоліки такого стеку:

1) Результати обрахунків залежать від браузера в якому виконується програма. Також можливі різні результати навіть у браузерів на одному движку, але різних версій

Так як у кожного браузеру є свої особливості роботи JS – потрібно зробити код максимально кросбраузерним та кросплатформеним. Для зборки проектів використовується Gulp[38] у поєднанні з Babel[37], який переганяє код під старі стандарти, а отже проект зможе запуститись і максимально корректно працювати як і в останній версіїї Google Chrome, так і в Internet Explorer.

3.2 Огляд та тестування програмного забезпечення

В основі роботи програмного забезпечення ієрархічної системи координаційного управління лежать алгоритми описані у другому розділі даної роботи. А саме алгоритм припинення обрахування по бажаному відхиленню а також оптимізації за алгоритмом Монте-Карло[47].

Стартовий екран, що відображений на рисунку 3.1, просить у користувача ввести розмір матриці для побудови графу.

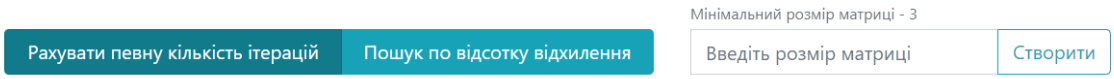


Рисунок 3.1 – Стартовий екран програми

Так як актуальність обрахунків стає актуальна для графу з мінімальним розміром у 3 вершини – програма видає помилку з допоміжним текстом для користувача (рис.3.2).

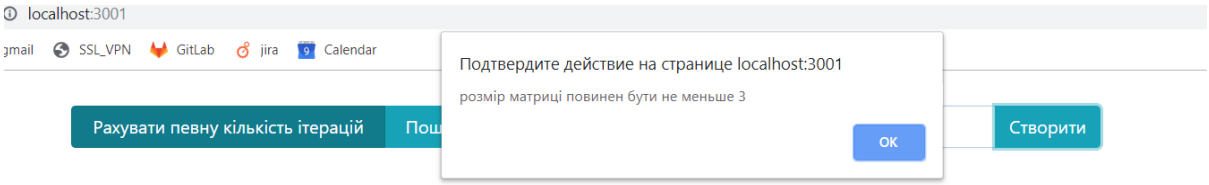


Рисунок 3.2 – Виведення помилки при малому розмірі матриці

Коли користувач введе задовольняючий нас розмір матриці і натисне кнопку створити – програма сама побудує матрицю розміром m*n, де m=n=введеному числу (рис 3.3) . А також відкриє поля для вводу(рис 3.4) обов’язкових коефіцієнтів впливу, що потрібні для обрахування найкращого у Out.

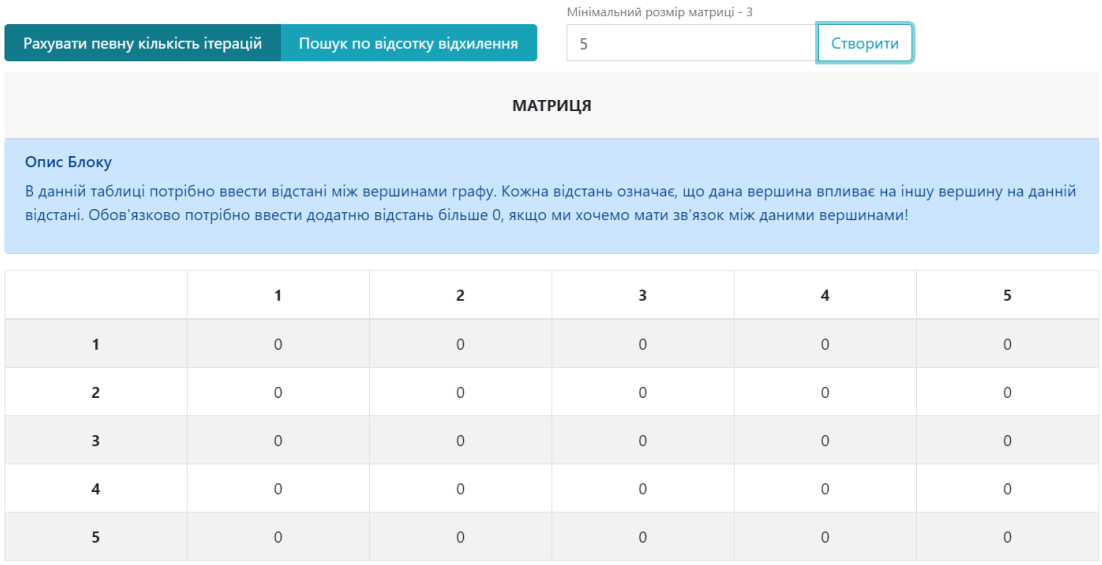


Рисунок 3.3 – Виведення матриці розміром m*n на екран користувача

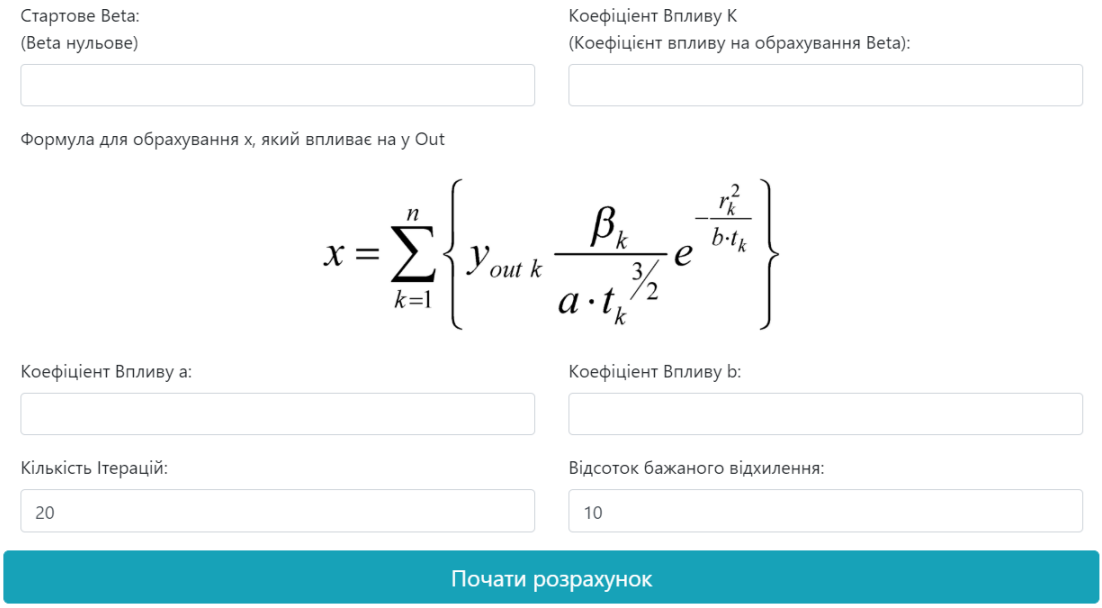


Рисунок 3.4 – Виведення полів для вводу коефіцієнтів на екран користувача

Було вирішено що матриця суміжності і матриця для вводу відстаней між точками – буде одна таблиця, в яку вводяться відстані між вершинами, що в свою чергу дає зрозуміти, що зв’язок між точками є . Це більше зручніше для усунення додаткових помилок, як було у прототипній версії

зображеній на рисунку 3.5 . І звичайно – програма не запуститься, якщо якась з відстаней не введена.

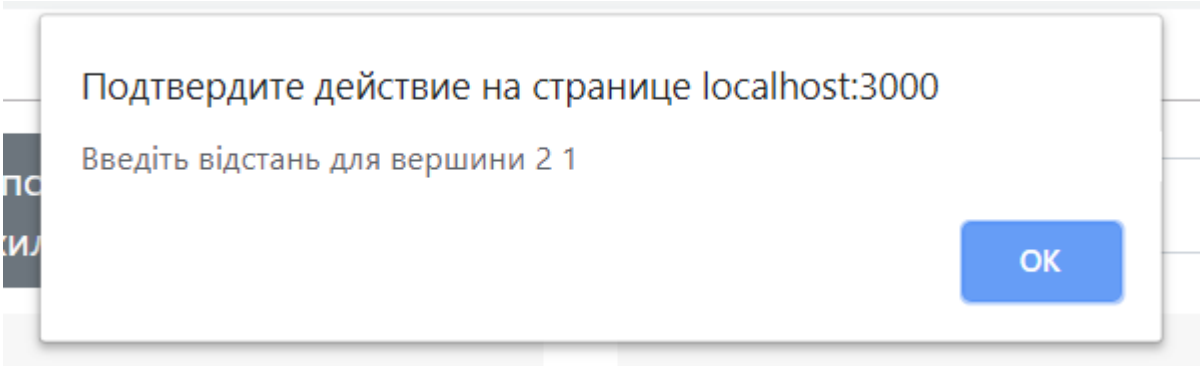


Рисунок 3.6 – Виведення полів у яких не вказана відстань

Також важливим питанням було введення коефіцієнтів для обрахування оптимізації. Є перевірка на введення даних у обов'язкові поля (рис 3.7) і програма також не запустить обрахунки, поки не введені всі дані.

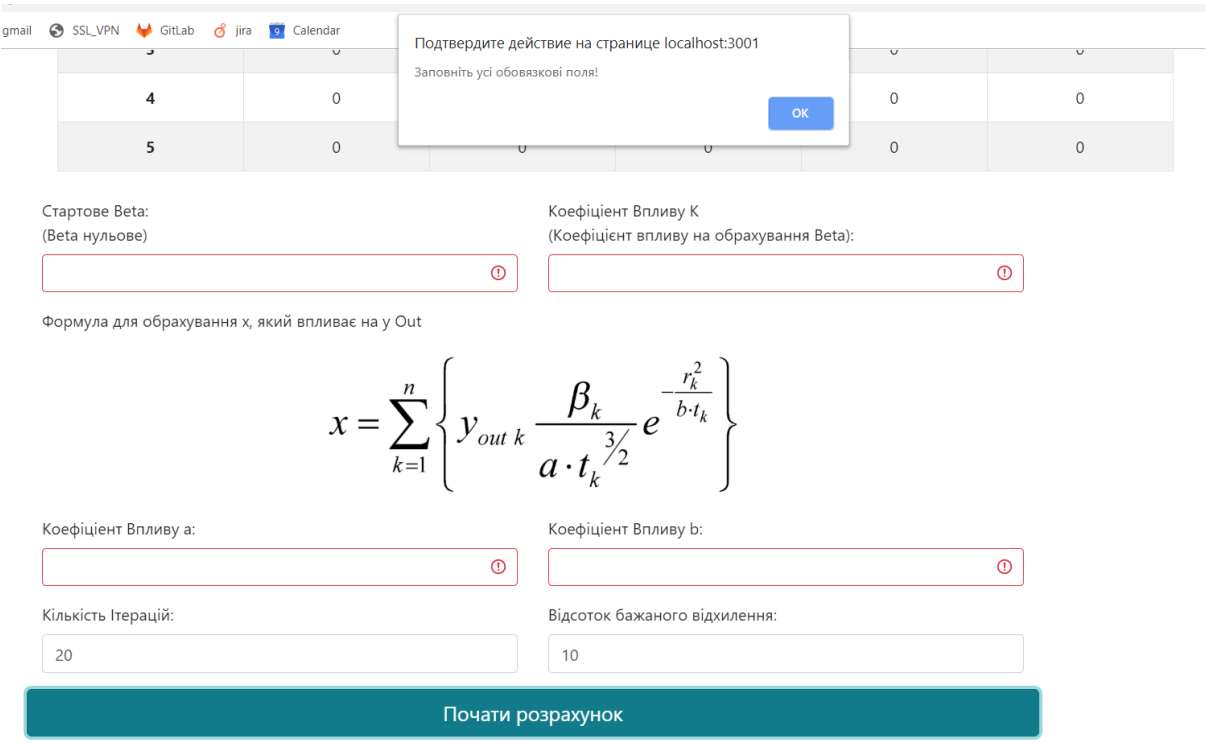


Рисунок 3.7 – Виведення помилки для обов'язкових полів

Якщо всі данні введені вірно – програма почне обрахунки. Після обрахунків всі дані виводяться у нову таблицю (рис 3.8). Ця таблиця містить у собі найкращий результат у Out, а також всі Бета, Бета нульове, а також у Out у всіх вершинах на ітерації з найкращим показником у Out.

РЕЗУЛЬТАТИ

Максимальний yOut : 0.13

	Beta	BetaZero	yOut
1	1.6171650000000004	1.7465382000000005	0
2	1.6171649888429676	1.7641800000000003	0.35393189389382435
3	2.1961500000000003	2.371842000000001	1.6140287024243123
4	1.6171650000000006	1.7465382000000005	0.0962489496470899
5	2.9526016666666677	3.543122000000002	0.12940136563664312

Рисунок 3.8 – Виведення таблиці з результатами

Так як дані у нас обраховуються динамічно, а також мають рандомний вплив – результати можуть бути різні, тому при наступному запуску обрахунку – дані оновлюються. Також ми можемо змінити лише один показник, (відстань, якийсь з коефіцієнтів і інше) – ми також отримаємо новий результат і оновлену таблицю результатів.

Сама програма має два режима, «Обрахунок по кількості ітерацій» і «Пошук по відсотку відхилення». За замовчуванням у нас обраний режим «Обрахунок по кількості ітерацій» з 20 ітераціями. Користувач може змінити кількість ітерацій у полі «Кількість Ітерацій:» у блоці з коефіцієнтами.

У верхній частині стартового екрану (рис 3.1) є дві кнопки перемикача режимів. Там ми можемо змінити на «Пошук по відсотку відхилення», по замовчуванню у нас бажаний відсоток відхилення = 10%, але ми можемо цей показник змінити у полі «Відсоток бажаного відхилення:» у блоці з коефіцієнтами.

Загальний інтерфейс усієї програми зображений на рисунку 3.9.

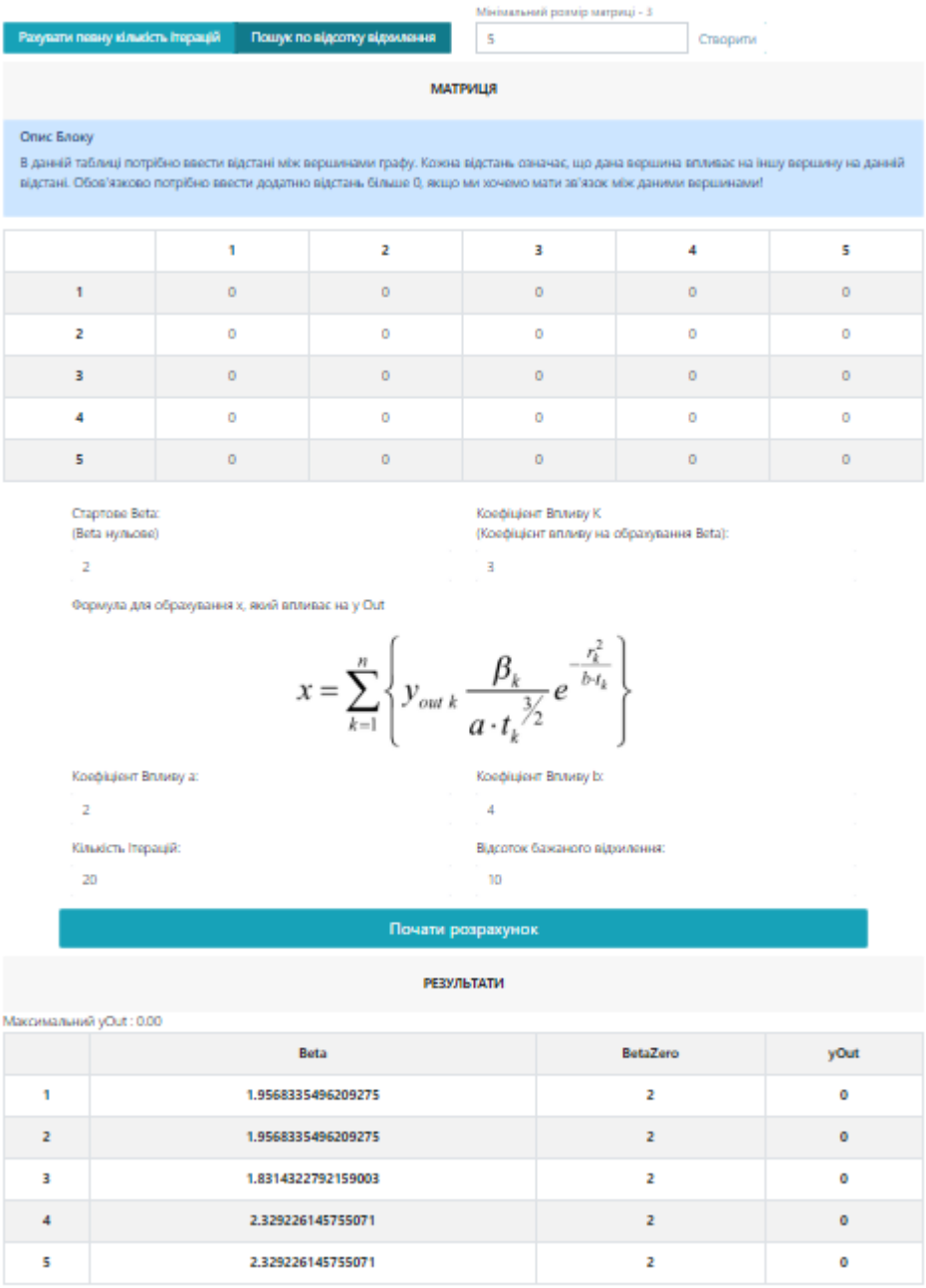


Рисунок 3.9 – Виведення таблиці з результатами

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

Вся логіка для коректної та комфортної роботи користувача описана у розділі «3.2 Огляд та тестування програмного забезпечення». Так як сказано вище у нас є перевірки на можливі помилки користувача, такі як пропуск вводу відстані між вершинами чи заповнення обов’язкових полів.

У деяких полів введені значення за замовчуванням, але для більш комфортної роботи та точного результату – їх краще підправити перед запуском обрахунків.

Для запуску програми потрібно мати установлений браузер на ПК. Якщо корисутвач має браузер і програма не знаходиться у мережі потрібно зайти в корінь папки проекті, знайти папку inline-build і запустити з неї файл index.html. Далі програма запуститься у браузері по замовчуванням.

3.3.2 Інструкція програміста

Першим ділом у програміста на ПК повинні бути глоабльно установлені Node та npm [50]. Для встановлення потрібних пакетів для розробки та зборки проекту.

Бажана версія Node.js - v10.17.0.[46]

Також потрібний глобально установлений Gulp, так як ми його використовуємо для перегону js ES6 у ES5, для коректної роботи у всіх браузерах.

Проект створеного програмного забезпечення зображено на рисунку 3.10.

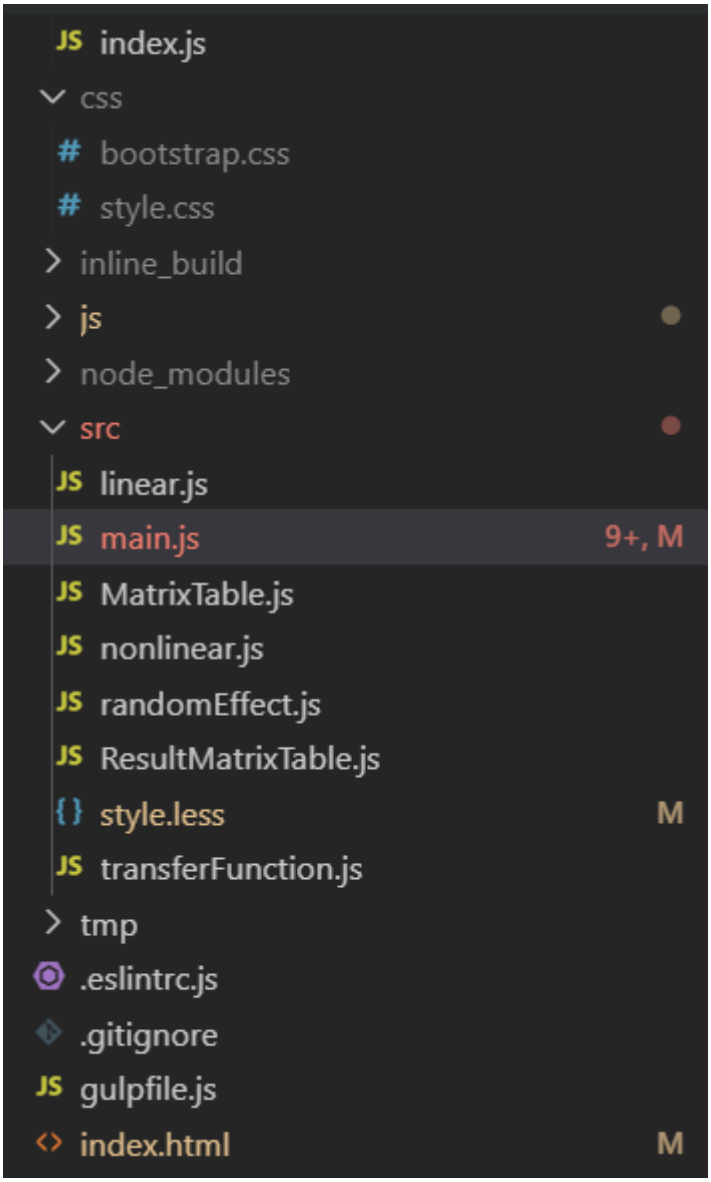


Рисунок 3.10 – Дерево проекту

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

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

Файл constants/index.js – файл з усіма глобальними константами

Скрипт src/transferFunction.js – обраховує результат транспортної функції

Скрипт src/getRandom.js – генерує випадковий вплив на Бета нульове

Клас src/MatrixTable.js – приймає розмір бажаної матриці, блок в який буде рендеритись табличка і назву таблички. Рендерить матрицю для вводу даних

Клас src/ResultMatrixTable.js – отримує об’єкт з даними і рендерить табличку з результатами.

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

npm install – встановить всі необхідні залежності для проекта.

gulp start –запуск проекту в режимі розробника.

Команда npm inline-build збере production версію проекта, де буде один файл, inline-build/index.html в якому будуть всі файли , що підключені у index.htmlз аттрибутом inline.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки програмного забезпечення системи ієрархічного 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	Технічні та споживчі властивості	Технічні та споживчі властивості	Технічні та споживчі властивост	Технічні та споживчі властивості

	продукту значно гірші, ніж в аналогів	продукту трохи гірші, ніж в аналогів	продукту на рівні аналогів	і продукту трохи кращі, ніж в аналогів	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	Термін реалізації ідеї від 3-х до 5-ти років.	Термін реалізації ідеї менше 3-х років. Термін	Термін реалізації ідеї менше 3-х років. Термін окупності

		інвестицій більше 10-ти років	Термін окупності інвестицій більше 5-ти років	окупності інвестицій від 3-х до 5-ти років	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 ієрархічних 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 M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

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

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

$В_і$ – 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:

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

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

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:

$$B_{заг} = \frac{B}{\alpha} \tag{4.8}$$

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

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

$$ЗВ = \frac{B}{\beta}, \tag{4.9}$$

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.

ВИСНОВОК

В ході виконання магістерської кваліфікаційної роботи розроблено програмне забезпечення ієрархічної системи координаційного управління.

В першому розділі був проведений детальний ієрархічних систем управління. А також розглянуто інших реальних систем і готових рішень на реальних прикладах.

Другий розділ присвячено розробці алгоритмів та uml-діаграм для ієрархічних систем координаційного управління. Розроблений алгоритм закінчення обрахунків по бажаному відсотку відхилення, а також алгоритми оптимізації вихідних даних на основі алгоритму Монте Карло. Також тут розроблено та описано uml-діаграми.

Третій розділ було розроблено програмне забезпечення, а саме логіка робота по обраним і розробленим алгоритмам, а також максимально простий у розумінні інтерфейс. Результат тестування видав позитивні показники відносно бажаних отриманих даних. У кінці розділу розроблено документацію для користувача та програміста, задля легко старту і коректної подальшої роботи з програмою.

Четвертий розділ присвячено економічному обґрунтуванню розробки. Розраховано загальні витрати на виконання всієї роботи, а також прогнозування загалних витрат на виконання та впровадження. Також було спрогнозовано комерційні ефекти від реалізації результатів розробки.

Отже, можна зробити висновок, що дане програмне забезпечення можна використовувати для координаційного управління ієрархічними системами.

СПИСОК ЛІТЕРАТУРИ

1. Посилання на рисунок [Електронний ресурс] – Режим доступу: https://www.hisour.com/ru/hierarchical-control-system-43222/
2. Коекин А. И. Оптимизация надежности и структуры иерархических систем управления. «Автоматика и телемеханика», 1965, т. 26
3. «A Multiagent System for Hierarchical Control and Monitoring», Journal of Universal Computer Science, vol. 15, no. 13 (2009), 2485-2505 submitted: 31/10/08, accepted: 13/6/09, appeared: 1/7/09 © J.UCS
4. Посилання на рисунок [Електронний ресурс] – Режим доступу: https://helpiks.org/6-20189.html
5. Стефані Е.П. «Основы построения АСУ ТП», М.: Энергоиздат, 1982. — 352 с,
6. Владзиевский А.П. «Устройство автоматических линий»
7. Посилання на рисунок [Електронний ресурс] – Режим доступу: http://scask.ru/f_book_kiber1.php?id=517
8. Куликовський Р. «Сложные системы управления». К., 1966
9. Посилання на рисунок [Електронний ресурс] – Режим доступу: https://mash-xxl.info/info/421676/
10. Ковшов А.Н «Технология машиностроения», 1987
11. Посилання на рисунок [Електронний ресурс] – Режим доступу: https://mash-xxl.info/info/76888
12. Смехов А.А. «Автоматизированые склады», 1979, 288с.
13. Инженерный справочник по космической технике. Изд. 2-е, перераб. и доп. Под ред. А В, Солодова, М., Воениздат, 1977, 430 с с НЛ
14. Жимерин Д.Г. «Технические средства управления энергетическими системами» , 1978, 288с.
15. Information about MIR computers [Електронний ресурс] – Режим доступу: http://www.suomentietokonemuseo.fi/vanhat/eng/mir2_eng.htm [Eng]

16. Л. Я. Карпман, О. А. Ющенко. Енциклопедія кібернетики, т. 1-2. К., 1973.
17. Unicon [Електронний ресурс] – Режим доступу: https://www.micron.com/ [Eng]
18. Ю.К. Машунин «Методы и модели векторной оптимизации», 1986
19. Тюрин М.Е., Макаров А.А. «Анализ данных на компьютере»
20. Genetich Algorithm [Електронний ресурс] – Режим доступу: https://www.micron.com/ [Eng]
21. Sadeghi, Javad; Sadeghi, Saeid; Niaki, Seyed Taghi Akhavan (10 July 2014). "Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: An improved particle swarm optimization algorithm". *Information Sciences*. 272: 126–144.
22. Akbari, Ziarati (2010). "A multilevel evolutionary algorithm for optimizing numerical functions" IJIEC 2 (2011): 419–430
23. Graph (discrete mathematics) [Електронний ресурс] / Wikipedia – 2009. – Режим доступу до ресурсу https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
24. 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.
25. Hameed S On Co-regular Signed Graphs // Australasian Journal of Combinatorics. 2014. P. 40-50.
26. Стефані Е.П. «Основы построения АСУ ТП», М.: Энергоиздат, 1982. — 352 с,
27. Algorithm Monte Carlo [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/an-overview-of-monte-carlo-methods-675384eb1694 [Eng]
28. Metropolis, N. (1987). "The beginning of the Monte Carlo method"

29. J.A. Bondy, U.S.R. Murty (1976). Graph Theory with Applications. Elsevier/North-Holland. c. 264 c. ISBN 0-444-19451-7.
30. Ruohonen K. Graph theory / K Ruohonen – USA: TUT, 2013. – 114 p.
31. 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.
32. Kamel M., Scoglio C., Easton T: Optimal topology design for overlay networks //. Lectures Notes in Computer Science, Springer, Vol. 4479/ 2007, p. 714–725.
33. Paul E. Black, "associative array", in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed. 17 December 2004.
34. "JavaScript". *Collins English Dictionary – Complete & Unabridged 2012 Digital Edition. William Collins Sons & Co. 2012*. Retrieved 21 August 2015.
35. User friendly UI. [Електронний ресурс] – Режим доступу: https://techterms.com/definition/user-friendly
36. User friendly UI. [Електронний ресурс] – Режим доступу: https://developers.google.com/web/tools/chrome-devtools
37. Babel. [Електронний ресурс] – Режим доступу: https://babeljs.io/
38. Gulp. [Електронний ресурс] – Режим доступу: https://gulpjs.com/
39. Thompson J. Future Perspectives: High-Performance Computing // Statistics for Bioinformatics Vol. 10, Issue 4. 2016. P. 33-70.
40. Mehrdad A.Mizani. Chapter 17 - Cloud-Based Computing // Key Advances in Clinical Informatics. 2017. P. 239-255.
41. М. А. Бучакова. Координация в управлении: теоретические подходы. Научный вестник Омской академии МВД России № 2(33), 2009.С.4-7.

42. Бурков, Владимир Novikov, Dmitry. (2009). Active systems theory (history of development). Problemy Upravleniya. 3. 29-35.
43. Chang A.S.etal. Coordination Needs and Performance for Manufacturing Process Improvement Projects. In: Advanced Materials Research, Vols. 311-313, pp. 2239-2244, (2011)
44. Woodside C. M., Monforton G.G. Fast Allocation of Processes in Distributed and Parallel Systems. In: IEEE Transactions on parallel and distributed systems 4(2), 164-174 (1993)
45. Ladanyuk A.P., Shumyhay D.A., Boyko R.O. System task coordination http://dspace.nuft.edu.ua/jspui/bitstream/123456789/4444/1/Sh_3.pdf. (2015)
46. Node.js. [Електронний ресурс] – Режим доступу: https://nodejs.org/en/
47. An Overview of Monte Carlo Methods [Електронний ресурс] / Medium – 2018. – Режим доступу до ресурсу: https://towardsdatascience.com/an-overview-of-monte-carlo-methods-675384eb1694
48. 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/
49. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих / А. Бхаргава – Питер: Питер, 2016. – 288
50. npm. [Електронний ресурс] – Режим доступу: https://www.npmjs.com/
51. 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 ієрархічної системи координаційного управління. Частина 1. Ієрархічна система

Студент групи 2АКІТ-18м Наумчук Д.О.

“___” _____ 2019 р.

Керівник к.т.н., доцент Юхимчук М. С.

“___” _____ 2019 р.

Вінниця 2019

1. Назва та галузь застосування

1.1. Назва – Інформаційна технологія ієрархічного 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 ієрархічному координаційному управлінні.

4. Вихідні дані для проведення розробки.

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

1. Коекин А. И. Оптимизация надежности и структуры иерархических систем управления. «Автоматика и телемеханика», 1965, т. 26

2. «A Multiagent System for Hierarchical Control and Monitoring», Journal of Universal Computer Science, vol. 15, no. 13 (2009), 2485-2505 submitted: 31/10/08, accepted: 13/6/09, appeared: 1/7/09 © J.UCS

3. Metropolis, N. (1987). "The beginning of the Monte Carlo method"

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- Введення відстаней та зв'язків між вершинам графу
- вибір режиму оптимізації
- Налаштування параметрів
- Оптимізація методом Монте-Карло

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:

- Постановка задачі «12 » 09. 2019 р.
- Аналіз предметної області «15 » 09. 2019 р.
- Розробка алгоритмів «3 » 10. 2019 р.
- UML-діаграма послідовності. «22 » 09. 2019 р.
- UML-діаграма варіантів використання «22 » 09. 2019 р.
- Розробка програмного забезпечення «03 » 12. 2019 р.
- Результат оптимізації «05 » 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

```
/* eslint-disable no-await-in-
loop */
/* eslint-disable guard-for-in */
import {
  pow, e,
} from 'mathjs';
import {
  kofeicientZ,
} from '../constants';
import transferFunction from './tra
nsferFunction';
import getRandom from './randomEffe
ct';
import MatrixTable from './MatrixTa
ble';
import ResultMatrixTable from './Re
sultMatrixTable';

//  ========== graphTopsObject SCHE
MA ===========
// [point_number]: <object>{
// V: <number>
// beta: <number>
// // betaZero: <object>{
// //// newBetaZero: <number>,
// //// oldBetaZero: <number>,
// // },
// distance: <object>{
// //// [point_number]: <number>
// }
// points: <array>[ <[point_number]
> ],
// x: <number>,
// yOut: <number>
// }

// об'єкт, який містить у собі імен
а прапорів для актуального значення
 Бета нульового на кожній ітерації
const newBetaKeys = {
  newBetaZeroValue: 'newBetaZero',
  oldBetaZeroValue: 'oldBetaZero',
};

// об'єкт, який фістить у собі прап
ори для вибору актуального режиму
const modes = {
  withLimit: 'withLimit',
  whileResult: 'whileResult',
};

const { withLimit, whileResult } =
modes;
const { newBetaZeroValue, oldBetaZe
roValue } = newBetaKeys;

const MAIN_MATRIX_DEXRIPTION = `
  В данній таблиці потрібно ввести
відстані між вершинами графу.
  Кожна відстань означає, що дана в
ершина впливає на іншу вершину на д
анній відстані.
  Обов'язково потрібно ввести додат
ню відстань більше 0, якщо ми хочем
о мати зв'язок між даними вершинами
!
`;

window.onload = () => {
  const canSearchBestResultLoop = t
rue;
  const CREATE_MATRIX_BTN = documen
t.querySelector('#create_matrix_btn
_js'); // кнопка на створення матри
ці
  const START_PROGRESS_BTN = docume
nt.querySelector('#start_progress_b
tn_js');
  const TOGGLE_CHANGE_MODE_BTNS = d
ocument.querySelectorAll('#toggle_m
ode_js input');
  const TOGGLE_CHANGE_MODE_LABELS =
 document.querySelectorAll('#toggle
_mode_js label');
  let resultsCOrreletionControlArr
= []; // в режимі пошуку відносно в
ідостку бажаного відхилення
  let bestYoutResultObj = null; //
об'єкт, який зберігає у собі копію
показників вершин графу при найкращ
ому yOut

  let MAIN_MATRIX = null; // матриц
я в DOM
```

```
  let graphTopsObject = {}; // об'єкт що заповнюється з матриць і використовується для всіх обрахувань
  let currentCountMode = withLimit;
  let addRandomEffect = false;
  let betaFlag = newBetaZeroValue; // newBetaZero or oldBetaZero keys
  let prevResult = 0; // змінна, яка зберігає результат Sigma з минулої ітерації дял

  let constantsObj = {
    startBeta: null,
    kofeicientK: null,
    kofeicientA: null,
    kofeicientB: null,
    loopCounterLimit: 20,
    desiredPercentage: 0.10, // бажаний відсоток відхилення
  };

  const loader = {
    item: document.querySelector('#preloader'),
    show() {
      this.item.setAttribute('data-show', true);
    },
    hide() {
      this.item.setAttribute('data-show', false);
    },
  };

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

  // підраховуємо X для конкретної вершини для подальшого вирахування yOut
  // формула X:
  // сума всіх yOut сусідніх вершин перемножених на Бета сусідніх вершин і інші коефіцієнти які загальні для всіх
  // формула в рядку де обраховується result
  const countGraphTopsX = ({ graphTop }) => new Promise((resolve) => {
    resolve(graphTop.points.reduce((accumulator, point) => {
      const currentTop = { ...graphTopsObject[point] };
      let yOut = currentTop.yOut || 1;
      if (isFinite(yOut)) yOut = 1;
      const distance = graphTop.distance[point];
      const { beta } = currentTop;
      const currentT = distance / kofeicientZ;
      const result = yOut * (beta / (constantsObj.kofeicientA * Number(pow(currentT, 3 / 2)) * Number(pow(e, Number(pow(distance, 2)) / (constantsObj.kofeicientB * currentT)))));
      return accumulator += result;
    }, 0));
  });


  // по черзі обраховується спочатку X для кожної вершини
  // потім yOut для кожної вершини
  // фкнція повертає Sigma, яка дорівнює yOut останньої вершини
  const searcForGraphXandYResults = async () => {
    let sigma = 0;
    // eslint-disable-next-line no-restricted-syntax
    for (const key in graphTopsObject) {
      const currentGraphTop = { ...graphTopsObject[key] };
```

```
      const newX = await countGraph
TopsX({ graphTop: currentGraphTop }
);
      const newYout = await countGr
aphTopsY({ graphTop: currentGraphTo
p, x: newX });
      graphTopsObject = {
        ...graphTopsObject,
        [key]: {
          ...currentGraphTop,
          x: newX,
          yOut: newYout,
        },

      };
      sigma = newYout;
    }
    return new Promise(resolve => r
esolve(sigma));
  };

  // обраховуємо Бета для конкретно
ї вершини
  // обраховує deltaD різниця попер
еднього Бета нульове  помножене на
рандосний вплив
  // ( Бета нульове з минулої ітера
ції яке ми вирішили використовувати
 (нове або старе) )
  // ( рандосний вплив - рандомне ч
исло 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 * delta
T(constanta)
  // ----------------------------
-----------
  //           Time1 + deltaT
  // newBeta підганяється до newBet
aZero циклом while поки різниця між
 бетами не буде меньше 10%
  // але є обмеження в 30 ітерацій,
 щоб не зайти в вічний цикл

  const countBeta = ({ beta = 0, V,
 currentBetaZero }) => new Promise(
async (resolve) => {
    const randomEffect = addRandomE
ffect ? getRandom() : 1;
    // const randomEffect = 1;
    const newBetaZero = currentBeta
Zero * randomEffect;
    let newBeta = beta;
    let transferFunctionResult = V;
    let iterattionCounter = 0;
    while ((Math.abs(newBeta - newB
etaZero)) / newBetaZero > 0.1 && it
erattionCounter < 30) {
      const deltaD = newBetaZero -
newBeta;
      transferFunctionResult = awai
t transferFunction({ deltaD, V: tra
nsferFunctionResult });
      newBeta = transferFunctionRes
ult * constantsObj.kofeicientK;
      iterattionCounter++;
    }

    resolve({ newBeta, newV: transf
erFunctionResult, newBetaZero });
  });

  // проходимось по всім вершинам г
рафа
  // обраховуємо новий Бета, бета н
ульове та V
  // заповнюємо цими даними об'єкт
з інформацією вершин
  const searcForGraphBetasResults =
 async () => {
    // eslint-disable-next-line no-
restricted-syntax
    for (const key in graphTopsObje
ct) {
      const currentGraphTop = { ...
graphTopsObject[key] };
      const beta = currentGraphTop.
beta || 0;
      const V = currentGraphTop.V |
| 1;
      const currentBetaZero = curre
ntGraphTop.betaZero ? currentGraphT
```

```
op.betaZero[betaFlag] : constantsOb
j.startBeta;
      const { newBeta, newV, newBet
aZero } = await countBeta({
        beta, V, currentBetaZero,
      });

      graphTopsObject = {
        ...graphTopsObject,
        [key]: {
          ...currentGraphTop,
          beta: newBeta,
          V: newV,
          betaZero: {
            newBetaZero,
            oldBetaZero: currentBet
aZero,
          },
        },
      };
    }
    return new Promise(resolve => r
esolve('object updated'));
  };
  // функція запускається при режим
і "Пошук по відсотку відхилення"
  // слідкує за кіількістю ітерацій
 з мінімальним відхиленням
  // якщо 3 ітерації відсоток зміни
 результату yOut змінюється меньше
ніж
  // бажаний відсоток відхилення -
програма закінчує свою роботу і вив
одить кращий результат
  const searchForSigmaWhileTrue = a
sync (counter = 0) => {
    counter++;

    if (canSearchBestResultLoop &&
counter < 1000) {
      await searcForGraphBetasResul
ts(); // count Y and Beta на кожній
 вершині
      const currentResult = await s
earcForGraphXandYResults();


      if (!bestYoutResultObj) {
        bestYoutResultObj = {
          graphTopsObject, prevResu
lt, betaFlag, counter,
        };
      }

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


      addRandomEffect = true;
      if (currentResult <= prevResu
lt) {
        const correletion = 1 - (cu
rrentResult / prevResult);
        if (correletion < constants
Obj.desiredPercentage) {
          resultsCOrreletionControl
Arr.push(graphTopsObject);
          if (resultsCOrreletionCon
trolArr.length === 3) {
            console.log(counter);

            console.log(graphTopsOb
ject);
            loader.hide();
            const resultMatrixDOM =
 new ResultMatrixTable({
              graphObject: bestYout
ResultObj.graphTopsObject,
              title: 'Результати',
              sigma: bestYoutResult
Obj.prevResult,
              parentBlock: '#result
_matrix_block',
              betaFlag: bestYoutRes
ultObj.betaFlag,
              counter: bestYoutResu
ltObj.counter,
            });
            console.log(prevResult)
;
```

```
            return true;
          }
        } else {
          resultsCOrreletionControl
Arr = [];
        }
        betaFlag = oldBetaZeroValue
;
        prevResult = currentResult;
        searchForSigmaWhileTrue(cou
nter);
      } else {
        resultsCOrreletionControlAr
r = [];
        betaFlag = newBetaZeroValue
;

        prevResult = currentResult;
        searchForSigmaWhileTrue(cou
nter);
      }

      // searchForSigmaWhileTrue();
    } else {
      console.log(counter);
      console.log(graphTopsObject);
      const resultMatrixDOM = new R
esultMatrixTable({
        graphObject: bestYoutResult
Obj.graphTopsObject,
        title: 'Результати',
        sigma: bestYoutResultObj.pr
evResult,
        parentBlock: '#result_matri
x_block',
        betaFlag: bestYoutResultObj
.betaFlag,
        counter: bestYoutResultObj.
counter,
      });
      loader.hide();

      console.log(prevResult);
    }
  };
  // функція запускається при режим
і "Рахувати певну кількість ітераці
й"
  // працює по заданому обмеженню к
ількості ітерацій
  // на останній ітерації воводитьс
я кращий результат з заданого відрі
зку ітерацііій

  const searchForSigmaLimitLoop = a
sync (counter = 0) => {
    if (counter < constantsObj.loop
CounterLimit) {
      await searcForGraphBetasResul
ts(); // count Y and Beta на кожній
 вершині
      const currentResult = await s
earcForGraphXandYResults();

      if (!bestYoutResultObj) {
        bestYoutResultObj = {
          graphTopsObject, prevResu
lt, betaFlag, counter,
        };
      }

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

      counter++;
      addRandomEffect = true;
      if (currentResult > prevResul
t) {
        betaFlag = newBetaZeroValue
;
      } else {
        betaFlag = oldBetaZeroValue
;
      }
```

```
      prevResult = currentResult;

      searchForSigmaLimitLoop(count
er);
    } else {
      console.log(graphTopsObject);
      const resultMatrix = new Resu
ltMatrixTable({
        graphObject: bestYoutResult
Obj.graphTopsObject,
        title: 'Результати',
        sigma: bestYoutResultObj.pr
evResult,
        parentBlock: '#result_matri
x_block',
        betaFlag: bestYoutResultObj
.betaFlag,
        counter: bestYoutResultObj.
counter,
      });
      loader.hide();
      console.log(prevResult);
    }
  };

  // функція будує таблиці по яким
після заповнення будуються матриці
і графи
  const createMatrixHandler = () =>
 {
    CREATE_MATRIX_BTN.addEventListe
ner('click', () => {
      const value = Number(document
.querySelector('#matrix_size_input_
js').value);
      if (value > 2) {
        MAIN_MATRIX = new MatrixTab
le(value, '#main_matrix_block', 'Ma
триця', MAIN_MATRIX_DEXRIPTION);
        document.querySelector('#se
ttings_block').setAttribute('data-
show', true);
      } else {
        alert('розмір матриці повин
ен бути не меньше 3');
        // matrixHelp.classList.tog
gle('show');
      }
    });
  };

  // функція бігає по таблиці матри
ці суміжності
  // і будує об'єкт з яким далі буд
емо працювати як графом

  const buildGraphObject = () => {
    graphTopsObject = {};
    console.log(graphTopsObject);
    const canCount = true;
    // матриця суміжності в якій ми
 маємо всі точки сполучень вершин
    const matrixValuesArr = documen
t.querySelectorAll('#main_matrix_bl
ock td');
    for (let i = 0; i < matrixValue
sArr.length; i++) {
      const graphTopNumber = Number
(matrixValuesArr[i].getAttribute('d
ata-i')); // вершина
      const graphLeftNumber = Numbe
r(matrixValuesArr[i].getAttribute('
data-
j')); // можлива сусідня вершина
      const value = Number(matrixVa
luesArr[i].innerHTML);
      // перша вершина графу в осно
ному не буде мати точок впливу,
      // але нам потрібна вона у сп
иску вершин для подалших обрахунків
 і впливів
      if (graphLeftNumber === graph
TopNumber && !graphTopsObject[graph
LeftNumber]) {
        graphTopsObject = {
          ...graphTopsObject,
          [graphLeftNumber]: {
            points: [],
            distance: {},
          }
          ,
        };
      }
```

```
      // перевіряємо значення у кож
ній точці матриці, якщо це значення
 1 - маємо вплив
      // і додаємо у вершину ( grap
hTopNumber ) вершину ( graphTopConn
ectNumber )
      // як вершина яка впливає
      // а також записуємо відстань
 між точками
      if (value > 0) {
        // мерджить данні в масиві
points
        const points = graphTopsObj
ect[graphLeftNumber]
          ? [...graphTopsObject[gra
phLeftNumber].points,
            graphTopNumber]
          : [graphTopNumber];

        // мерджить данні в масиві
distance
        const distance = graphTopsO
bject[graphLeftNumber]
          ? {
            ...graphTopsObject[grap
hLeftNumber].distance,
            [graphTopNumber]: value
,
          }
          : { [graphTopNumber]: val
ue };

        graphTopsObject = {
          ...graphTopsObject,
          [graphLeftNumber]: {
            points: [
              ...points,
            ],
            distance: {
              ...distance,
            },
          }
          ,
        };
      }
    }
    console.log(graphTopsObject);
    if (canCount) {
      bestYoutResultObj = null;
      loader.show();

      if (currentCountMode === with
Limit) {
        searchForSigmaLimitLoop();
      } else if (currentCountMode =
== whileResult) {
        searchForSigmaWhileTrue();
      }
    }
    // } else {
    //   throw new Error('матриця с
уміжності та матриця відстаней має
різні розміри');
    // }
  };

  // функція запуску оброки даних
  // перевіряє всі обов'язкові поля
  // якщо всі обов'язкові поля зада
ні - запускає  функцію побудови об'
єкту (buildGraphObject)
  const startWorkHandler = () => {
    START_PROGRESS_BTN.addEventList
ener('click', (e) => {
      e.preventDefault();

      let canBuildGraph = true;
      const inputs = document.query
SelectorAll('.input_data_js');

      inputs.forEach((element) => {
        const valueName = element.g
etAttribute('data-name');
        let value = Number(element.
value);
        if (valueName === 'desiredP
ercentage') value /= 100;
        if (value !== 0) {
          element.classList.remove(
'is-invalid');
          constantsObj = {
            ...constantsObj,
            [valueName]: value,
          };
        } else {
```

```
          element.classList.add('is
-invalid');
          canBuildGraph = false;
        }
      });
      console.log(constantsObj);
      if (canBuildGraph) buildGraph
Object();
      if (!canBuildGraph) alert('За
повніть усі обовязкові поля!');
      // buildGraphObject();
    });
  };
  // функція зміни режими обробки д
анних
  const toggleModeHandler = () => {
    TOGGLE_CHANGE_MODE_BTNS.forEach
((btn) => {
      btn.addEventListener('click',
 () => {
        currentCountMode = btn.getA
ttribute('data-state');
        TOGGLE_CHANGE_MODE_LABELS.f
orEach(element => element.classList
.remove('active'));
        btn.parentElement.classList
.add('active');
      });
    });
  };

  const startProcess = async () =>
{
    createMatrixHandler();
    startWorkHandler();
    toggleModeHandler();
    // searchForSigmaLimitLoop();
  };

  startProcess();
};
```

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 ІЄРАРХІЧОЇ СИСТЕМИ КООРДИНАЦІЙНОГО УПРАВЛІННЯ. ЧАСТИН 1. ІЄРАРХІЧНА СИСТЕМА

1. Постановка задачі
2. Аналіз предметної області
3. Розробка алгоритмів
4. UML-діаграма послідовності.
5. UML-діаграма варіантів використання
6. Розробка програмного забезпечення
7. Результат оптимізації

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

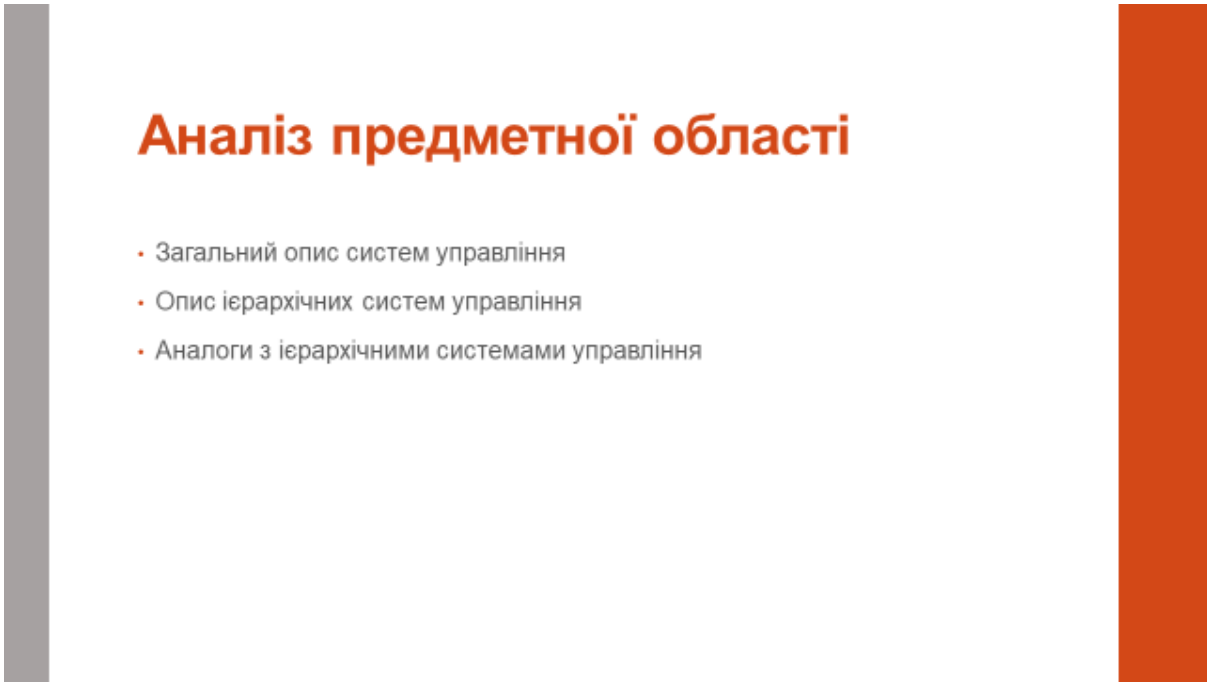
Вінниця 2019

Постановка задачі

Постановка задачі

- Розробити алгоритм і запрограмувати векторну оптимізацію.
- Запрограмувати алгоритм оптимізації за одним критерієм і обмеженням на параметри. Використати алгоритм Монте Карло
- Розробити програму, яка буде обраховувати вихідні параметри по вхідним, введеним з інтерфейсу користувачем. Припинення роботи програми по обмеженню

Аналіз предметної області



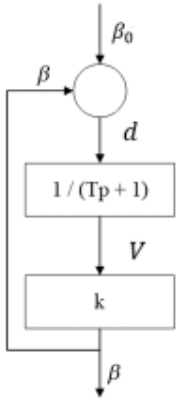
Розробка алгоритмів

РОЗРОБКА АЛГОРИТМІВ ТА UML-ДІАГРАМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- Розробка алгоритму оптимізації, що базується на оптимізації за одним критерієм і обмеженням на параметри
  - Оптимізація методом Монте-Карло

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

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



РОЗРОБКА АЛГОРИТМІВ ТА UML-ДІАГРАМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

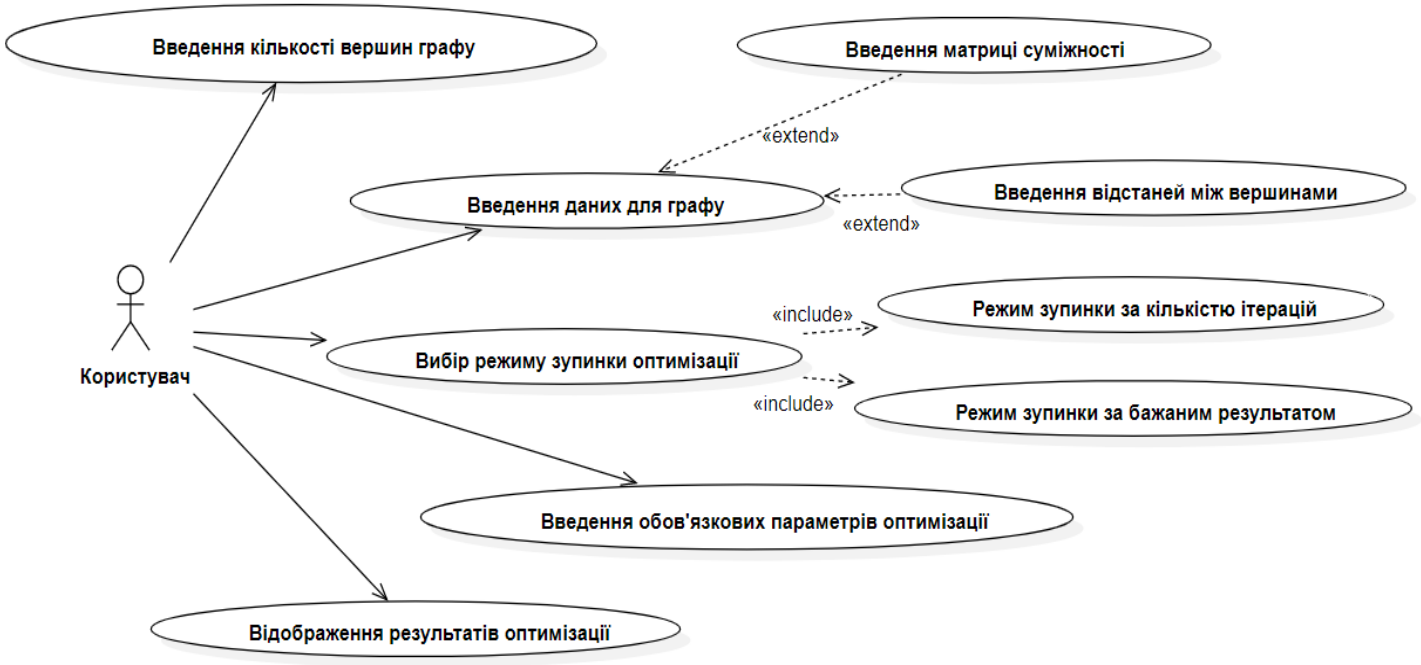
- Розробка методу знаходження впливу для кожної вершини

та методу оптимізації впливів між вершинами для

ієрархічних систем координаційного впливу

  - Припинення обрахування по бажаному відхиленню
  - Припинення обрахування по заданій кількості проходжень

UML діаграма варіантів використання

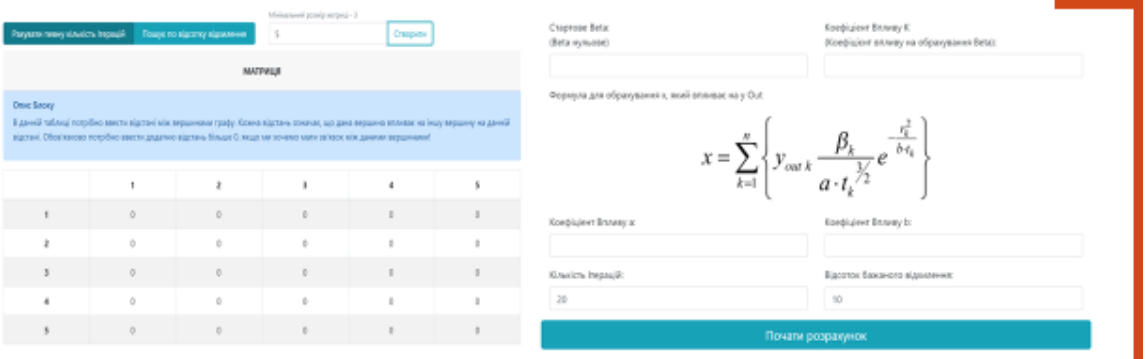


UML-діаграма послідовності.

Розробка програмного забезпечення

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- Екрани вводу даних



РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- Відловлення помилок

Результат оптимізації

Мінімальний розмір матриці - 3

Рахувати повну кількість ітерацій | Пошук по відсотку відхилення | 5 | Створити

МАТРИЦЯ

Опис Блоку

В даній таблиці потрібно ввести відстані між вершинами графу. Кожна відстань означає, що дана вершина впливає на іншу вершину на даній відстані. Обов'язково потрібно ввести додатню відстань більше 0, якщо ми хочемо мати зв'язок між даними вершинами!

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Стартове Beta:
(Beta нульове)
2

Коефіцієнт Впливу K
(Коефіцієнт впливу на обрахування Beta):
3

Формула для обрахування x, який впливає на y Out

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

Коефіцієнт Впливу a:
2

Коефіцієнт Впливу b:
4

Кількість Ітерацій:
20

Відсоток бажаного відхилення:
10

Почати розрахунок

РЕЗУЛЬТАТИ

Максимальний yOut : 0.00

	Beta	BetaZero	yOut
1	1.9568335496209275	2	0
2	1.9568335496209275	2	0
3	1.8314322792159003	2	0
4	2.329226145755071	2	0
5	2.329226145755071	2	0