Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра прикладної математики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Задача про призначення на тридольному графі з нечіткими вершинами
(тема)

Виконав:
здобувач _2_ року навчання, групи _САУм-24-1
Євген ОЩЕПКОВ
(Власне ім'я, ПРІЗВИЩЕ)
Спеціальність
124 Системний аналіз
(код і повна назва спеціальності)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма
Системний аналіз і управління
(повна назва освітньої програми)
Керівник доц. Ольга МАТВІЄНКО
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту

Завідувач кафедри ПМ _____ Максим СИДОРОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет _інформаційно-аналітичних технологій та менеджменту
Кафедра _прикладної математики
Рівень вищої освіти _другий (магістерський)
Спеціальність _124 Системний аналіз
(код і повна назва)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма _Системний аналіз і управління
(повна назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри _____
(підпис)
“ 10 ” листопада 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ощепкову Євгену Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Задача про призначення на тридольному графі з нечіткими вершинами

затверджена наказом по університету від 10 листопада 2025 р. № 1027 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 18 грудня 2025 р.

3. Вихідні дані до роботи математична модель задачі про призначення на тридольному графі з нечіткими вершинами

4. Перелік питань, що потрібно опрацювати в роботі
1. Системний аналіз предметної області
2. Вибір і обґрунтування методу розв’язання
3. Програмна реалізація
4. Результати обчислювального експерименту
5. Аналіз можливих застосувань

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

1. Актуальність теми роботи
2. Постановка задачі
3. Системний аналіз предметної області
4. Метод чисельного аналізу
5. Результати обчислювального експерименту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Підбір та вивчення технічної літератури за темою роботи	10 – 16 листопада 2025 р.	виконано
2	Вибір та обґрунтування методу	17 – 23 листопада 2025 р.	виконано
3	Розробка алгоритму і програми	24 – 30 листопада 2025 р.	виконано
4	Проведення аналітичних досліджень та розрахунків	01 – 07 грудня 2025 р.	виконано
5	Робота над текстом пояснювальної записки	08 – 17 грудня 2025 р.	виконано
6	Представлення роботи на рецензію в ЕК	18 грудня 2025 р.	виконано

Дата видачі завдання _10_ листопада 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Ольга МАТВІЄНКО
(підпис) (посада, Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка: 79 с., 9 табл., 34 рис., 1 дод., 30 джерел.

АЛГОРИТМ ЕДМОНДСА-КАРПА, ВІЗУАЛІЗАЦІЯ ДАНИХ, ГРАФІЧНИЙ ІНТЕРФЕЙС, ЗАДАЧА ПРО ПРИЗНАЧЕННЯ, МАКСИМАЛЬНИЙ ПОТІК, МЕТОД ФОРДА-ФАЛКЕРСОНА, НЕЧІТКА ЛОГІКА, НЕЧІТКІ МНОЖИНИ, ОПТИМІЗАЦІЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ, СИСТЕМА ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ, ТРИДОЛЬНИЙ ГРАФ.

Об’єкт дослідження – задача про призначення на тридольному графі з нечіткими вершинами.

Мета роботи – підвищення ефективності розподілу ресурсів у трикомпонентних системах шляхом дослідження математичної моделі, алгоритму та розробки програмного забезпечення для розв’язання задачі про призначення на тридольному графі з нечіткими вершинами.

Методи дослідження – методи системного аналізу, теорія графів, теорія нечітких множин, методи комбінаторної оптимізації та теорії потоків, зокрема алгоритми Форда-Фалкерсона та Едмондса-Карпа, а також об’єктно-орієнтоване програмування для практичної реалізації системи мовою Python.

У кваліфікаційній роботі проведено дослідження проблеми оптимального розподілу ресурсів у складних організаційних системах. Проаналізовано існуючі підходи до розв’язання задач про призначення та обґрунтовано доцільність використання нечіткої логіки для формалізації невизначених параметрів сумісності об’єктів.

Досліджено алгоритм пошуку максимального тривершинного поєднання, який базується на зведенні вихідної задачі до задачі про максимальний потік у транспортній мережі з ітеративною фільтрацією зв’язків за рівнем надійності. Розглянуто особливості застосування алгоритму Едмондса-Карпа для знаходження потоків у мережах з одиничними пропускними здатностями.

Результатом роботи є розроблене програмне забезпечення мовою Python з графічним інтерфейсом користувача. Програма дозволяє вводити нечіткі характеристики об’єктів, будувати граф з параметрами їх сумісності, виконувати пошук оптимального розв’язку та візуалізувати процес відсіювання ненадійних зв’язків.

Рекомендації щодо використання результатів роботи включають застосування розробленого програмного забезпечення як системи підтримки прийняття рішень при формуванні команд, розподілі технічних засобів та плануванні ресурсів. Також перспективним є використання результатів роботи з метою дослідження задачі про призначення з нечіткими вершинами на графах більшої розмірності.

Сфера застосування охоплює управління персоналом, логістику, служби надзвичайних ситуацій та проектний менеджмент.

Значимість роботи полягає у подальшому розвитку методів розв’язання комбінаторних задач оптимізації шляхом інтеграції класичних графових алгоритмів з методами нечіткої логіки, що дозволяє підвищити адекватність моделей при застосуванні в реальних умовах.

ABSTRACT

Introductory note: 79 pages, 9 tables, 34 figures, 1 appendix, 30 sources.

ASSIGNMENT PROBLEM, DECISION SUPPORT SYSTEM, EDMONDS-KARP ALGORITHM, FORD-FULKERSON METHOD, FUZZY LOGIC, FUZZY SETS, GUI, MAX FLOW, OPTIMIZATION, PYTHON, SOFTWARE IMPLEMENTATION, TRIPARTITE GRAPH, VISUALIZATION.

Object of research – the assignment problem on a tripartite graph with fuzzy vertices.

Purpose of work – to improve the efficiency of resource allocation in three-component systems by studying the mathematical model and algorithm, as well as by developing software for solving the assignment problem on a tripartite graph with fuzzy vertices.

Methods of research – methods of systems analysis, graph theory, fuzzy set theory, combinatorial optimization methods and flow theory, in particular the Ford–Fulkerson and Edmonds–Karp algorithms, as well as object-oriented programming for the practical implementation of the system in the Python programming language.

The qualification work investigates the problem of optimal resource allocation in complex organizational systems. Existing approaches to solving assignment problems are analyzed, and the expediency of using fuzzy logic to formalize uncertain parameters of object compatibility is substantiated.

An algorithm for finding the maximum three-vertex matching is researched, based on reducing the initial problem to the maximum flow problem in a transport network with iterative filtering of connections by reliability level. The specifics of applying the Edmonds-Karp algorithm for finding flows in unit capacity networks are considered.

The result of the work is software developed in Python with a graphical user interface. The program allows inputting fuzzy characteristics of objects, building a

compatibility graph, performing optimal solution search, and visualizing the process of filtering unreliable connections.

Recommendations for using the results include applying the developed software as a decision support system for team formation, technical equipment distribution, and resource planning.

The scope of application covers personnel management, logistics, emergency services, and project management.

The significance of the work lies in the further development of methods for solving combinatorial optimization problems by integrating classical graph algorithms with fuzzy logic methods, which allows increasing the adequacy of models when applied in real-world conditions.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

GUI – Graphical User Interface – графічний інтерфейс користувача;

IDEF0 – Integration Definition for Function Modeling – методологія функціонального моделювання;

DFD – Data Flow Diagram – діаграма потоків даних;

BFS – Breadth-First Search – пошук у ширину;

СППР – система підтримки прийняття рішень;

ТНЧ – трикутне нечітке число.

ВСТУП

Актуальність теми. Задачі оптимального розподілу ресурсів та призначень є невід'ємною складовою процесів управління в організаційних системах різної природи. Класична постановка задачі про призначення традиційно моделюється на дводольних графах. Проте в сучасних умовах ускладнення технологічних процесів та логістичних ланцюгів часто виникає необхідність враховувати взаємодію трьох та більше категорій об'єктів одночасно. Це вимагає застосування більш складних моделей, зокрема тридольних графів.

Особливістю реальних задач прийняття рішень є наявність невизначеності. Інформація про ефективність взаємодії об'єктів часто має нечіткий характер і базується на експертних оцінках. Детерміновані моделі ігнорують цю невизначеність, що призводить до отримання рішень, які є математично точними, але нестійкими або неадекватними на практиці. Тому актуальним є застосування апарату нечіткої логіки (Fuzzy Logic) [1, 2] для формалізації параметрів сумісності та дослідження методів пошуку рішень, що гарантують заданий рівень надійності системи.

Мета і завдання кваліфікаційної роботи. Метою роботи є підвищення ефективності розподілу ресурсів у трикомпонентних системах шляхом дослідження математичної моделі, алгоритму та розробки програмного забезпечення для розв'язання задачі про призначення на тридольному графі з нечіткими вершинами. Для досягнення поставленої мети необхідно виконати наступні завдання:

– провести огляд і аналіз сучасного стану задачі «про призначення на тридольному графі з нечіткими вершинами»;

– обрати метод дослідження поставленої задачі та дослідити алгоритм її розв'язання;

– самостійно попрацювати з літературою, провести дослідження, підібрати та опрацювати матеріали, що включає пошук та вивчення наукових статей, книг та інших джерел, які стосуються теми дослідження;

– застосовувати методи та інструменти системного аналізу для результативного та ефективного дослідження систем та процесів різної природи;

– виконати програмну реалізацію мовою Python дослідженого методу у вигляді додатку з графічним інтерфейсом;

– провести обчислювальний експеримент для перевірки коректності та ефективності роботи програми на прикладі задачі комплектування автономних рятувальних груп.

Об'єктом дослідження є задача про призначення на тридольному графі з нечіткими вершинами.

Предметом дослідження є математичні моделі та алгоритми розв'язання задачі про призначення на тридольних графах з використанням нечіткої логіки та теорії потоків у мережах.

Методи дослідження. У кваліфікаційній роботі використовуються методи системного аналізу, теорія графів, теорія нечітких множин, методи комбінаторної оптимізації та теорії потоків, зокрема алгоритми Форда-Фалкерсона та Едмондса-Карпа, а також об'єктно-орієнтоване програмування для практичної реалізації системи мовою Python.

Публікації. Результати, отримані у кваліфікаційній роботі, було представлено на 29-му Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» (м. Харків, 16-19 квітня 2025 р.) [3].

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Системний аналіз задачі про призначення на тридольному графі з нечіткими вершинами

1.1.1 Вербальна модель системи

Об’єкт аналізу – «Задача про призначення на тридольному графі з нечіткими вершинами».

Предмет аналізу – «Математичні моделі та методи розв'язання задачі про призначення на тридольному графі в умовах нечіткості».

Точка зору – дослідник.

Ціль – системний аналіз проблеми математичного моделювання та оптимізації задачі про призначення на тридольному графі з нечіткими вершинами.

Модель типу «чорна скриня» (рис. 1.1) показує взаємодію системи із зовнішнім середовищем, розглядаючи поведінку системи та її призначення.

Основні входи системи:

– множини об’єктів (модулі, зони, обладнання і т.д.);

– ступені сумісності об’єктів.

Основні виходи системи:

– оптимальне тривершинне поєднання;

– потужність отриманого поєднання;

– надійність отриманого поєднання.

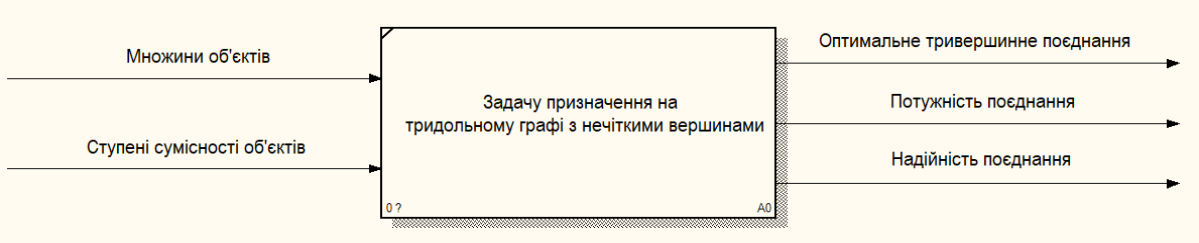

Рисунок 1.1 – Модель «чорна скриня»

1.1.2 Морфологічний опис системи

Морфологічний опис системи дозволяє розглянути основні елементи системи, їх функції та взаємозв’язки. Морфологічний аналіз допомагає виявити сильні та слабкі сторони системи, що дозволяє розробити стратегії для її вдосконалення. Для цього аналізу ми використовуємо модель зовнішнього середовища системи (рис. 1.2), яка відображає важливі компоненти та їхню взаємодію з задачею.

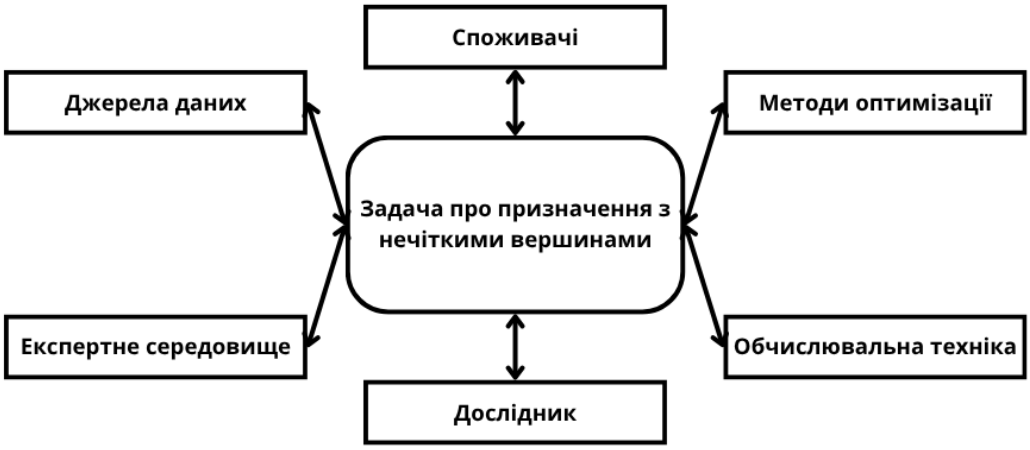

Рисунок 1.2 – Модель зовнішнього середовища системи

Споживачі формулюють вимоги до рішення та отримують кінцевий результат – оптимальний план призначення та його характеристики (потужність, надійність).

Джерела даних забезпечують систему вихідною інформацією: структура графу (опис множин об'єктів та можливих зв'язків), кількісні та якісні характеристики об'єктів.

Експертне середовище формує нечіткі оцінки для вершин і ребер та задає функції належності. Експертна інформація використовується для побудови нечіткої моделі системи.

Методи оптимізації включають теоретичні основи та алгоритмічні підходи, що використовуються для розв'язання задачі. Сюди відносяться теорія гра-

фів, теорія нечітких множин та методи знаходження максимального потоку.

Обчислювальна техніка представляє апаратне та програмне забезпечення, необхідне для реалізації алгоритмів, зберігання даних та виконання розрахунків.

1.1.3 Функціональна модель системи

Функціональна модель системи описує, як окремі компоненти системи взаємодіють для досягнення основних цілей. Вона дозволяє зрозуміти основні процеси, що відбуваються всередині системи, і як вони впливають на кінцевий результат.

Для означення вимог та функцій системи розробляється IDEF0 модель (рис. 1.3 – 1.6), яка відповідає вимогам та реалізує функції системи. IDEF0 може застосовуватися для аналізу функцій системи та відображення механізмів виконання цих функцій.

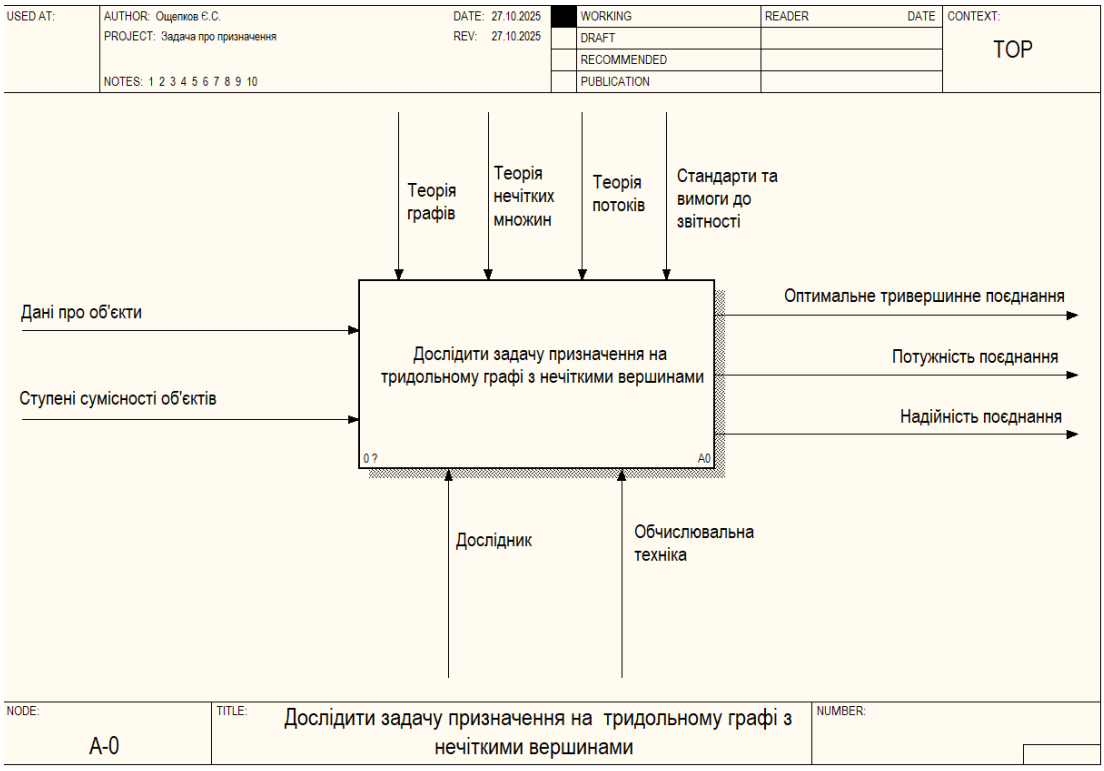

Рисунок 1.3 – Контекстна діаграма (рівень A-0)

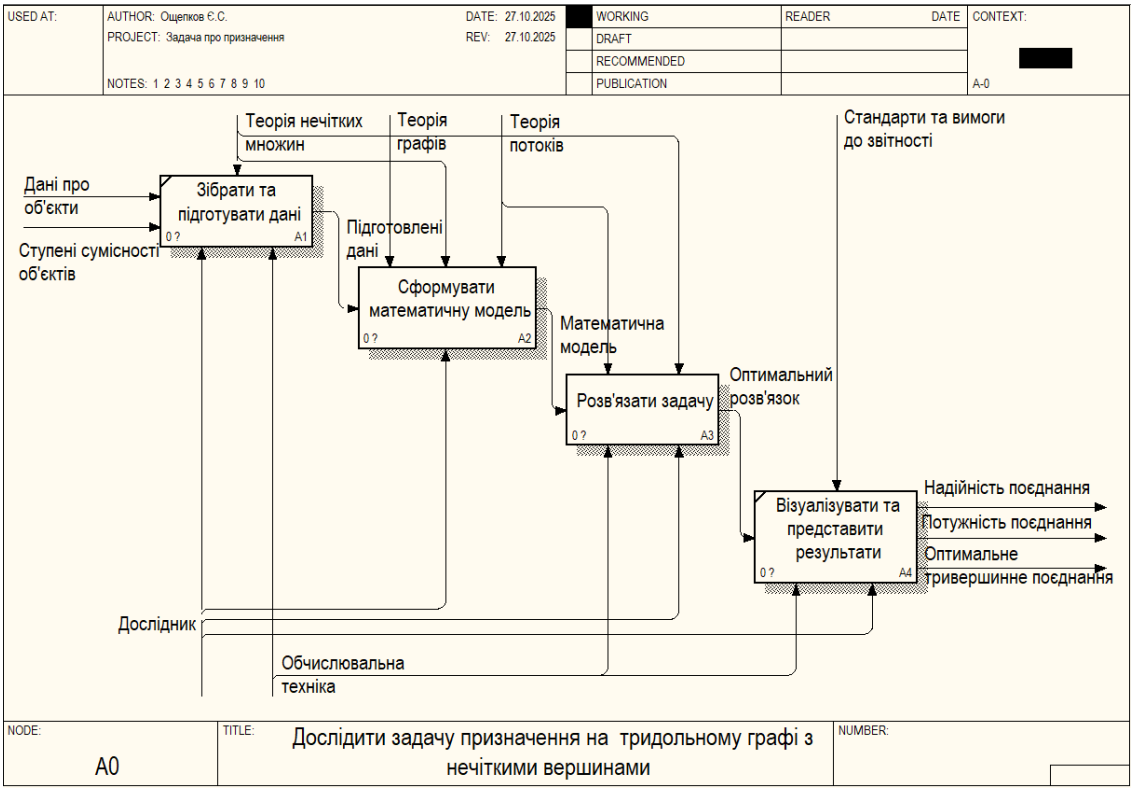

Рисунок 1.4 – Декомпозиція процесу «Дослідити задачу про призначення на тридольному графі з нечіткими вершинами»: (рівень A0)

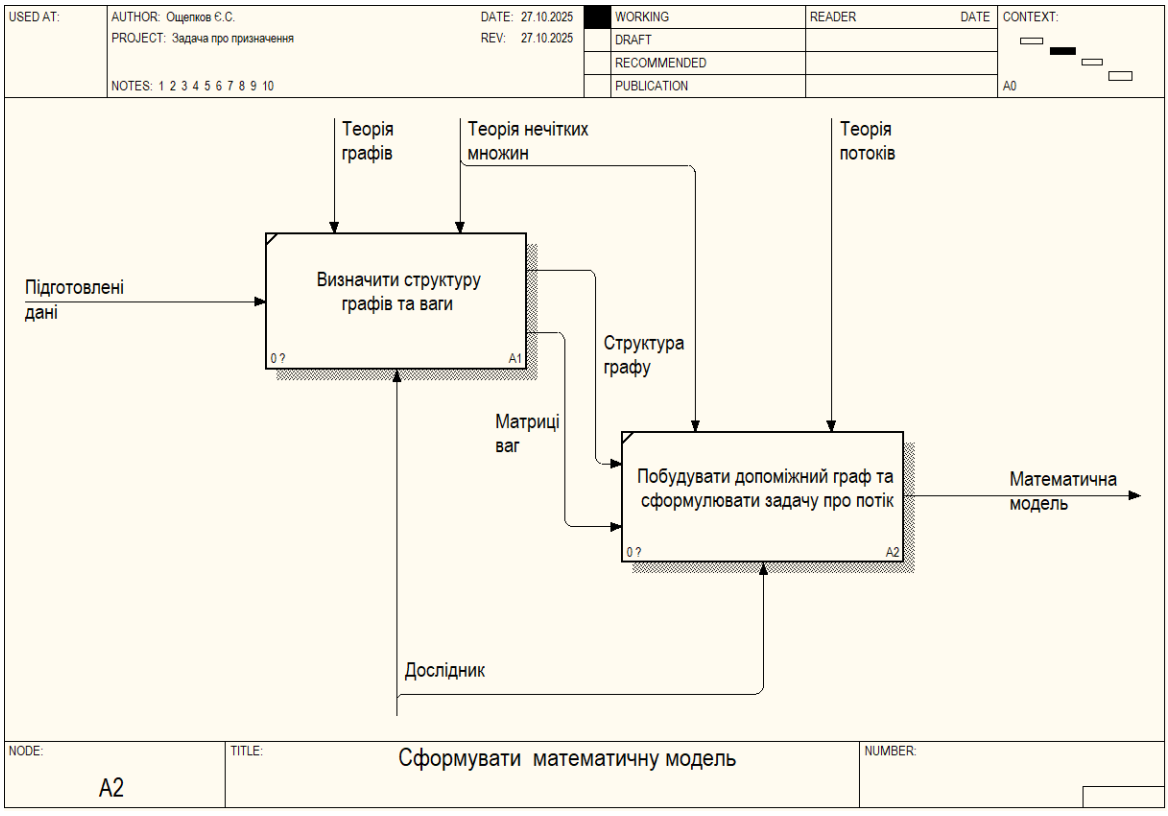

Рисунок 1.5 – Декомпозиція процесу «Сформувати математичну модель»: (рівень A2)

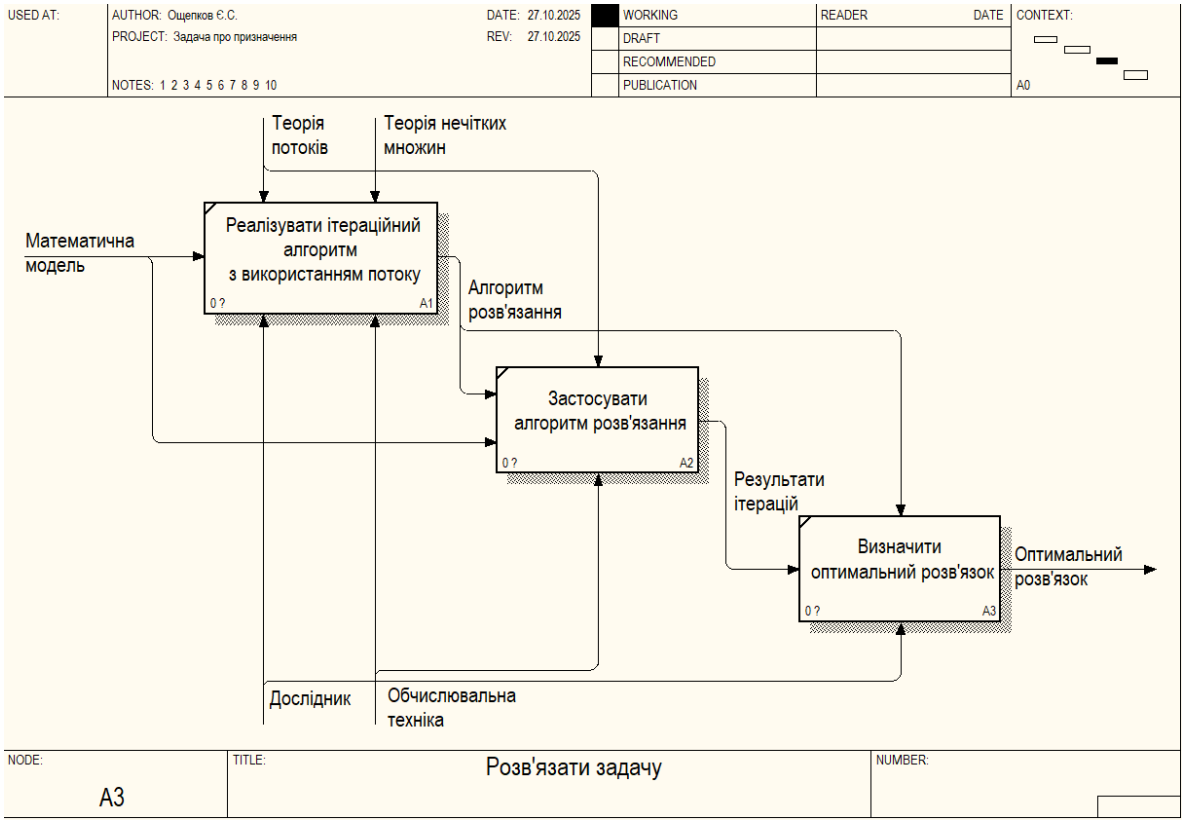

Рисунок 1.6 – Декомпозиція процесу «Розв'язати задачу»: (рівень A3)

1.1.4 Інформаційна модель

Інформаційна модель системи відображає зв’язки між її елементами у вигляді структур даних, звертаючи увагу на взаємозв’язки та склад потоків даних. Для побудови цієї моделі використовується методологія Data Flow Diagram (DFD), яка дозволяє візуалізувати потоки даних між різними компонентами системи. Методологія Data Flow Diagram (DFD) є надзвичайно корисною для візуалізації інформаційних потоків. Вона надає можливість побудувати діаграми, що демонструють, як дані проходять через систему, з яких джерел вони надходять, як обробляються та куди направляються. Вона застосовується додатково до моделі IDEF0 для чіткого відображення поточних операцій. На рисунку 1.7 наведена DFD-діаграма, а на рисунку 1.8 – декомпозиція.

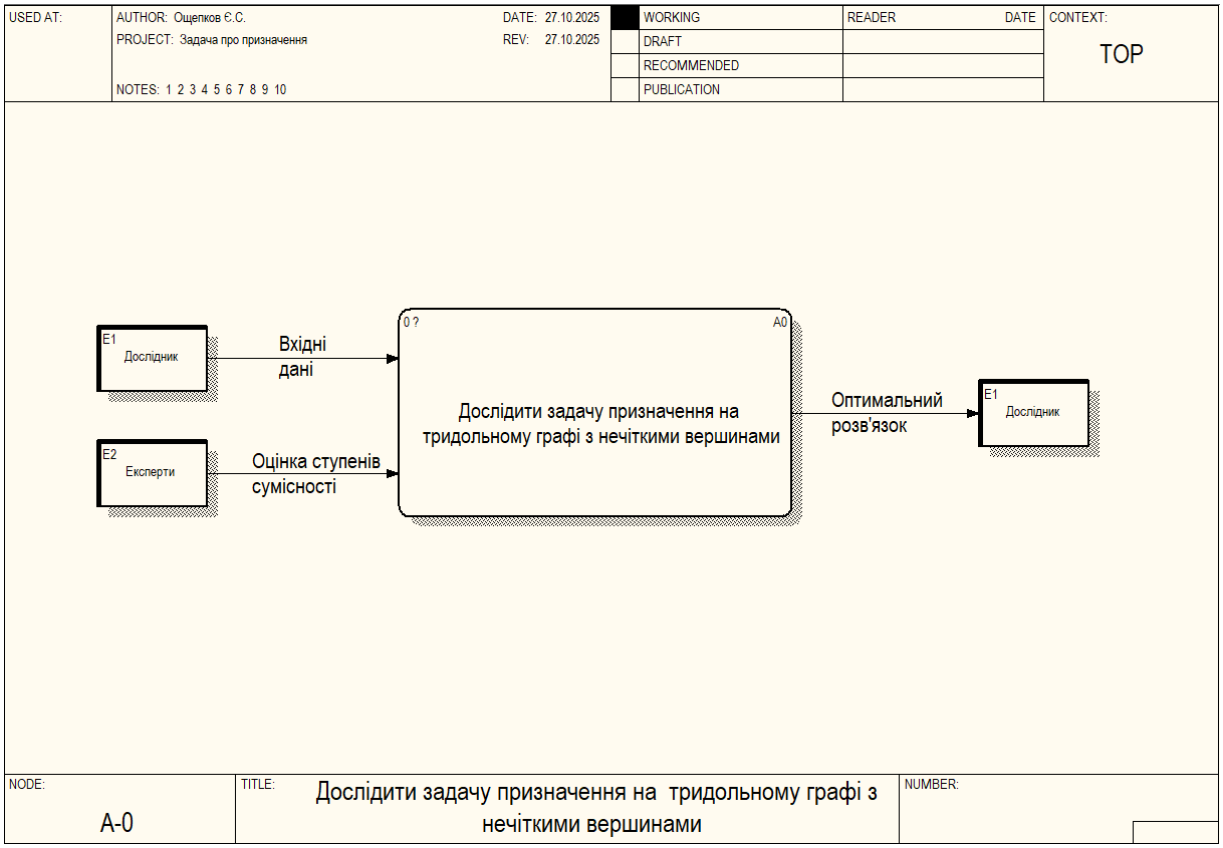

Рисунок 1.7 – DFD-Діаграма

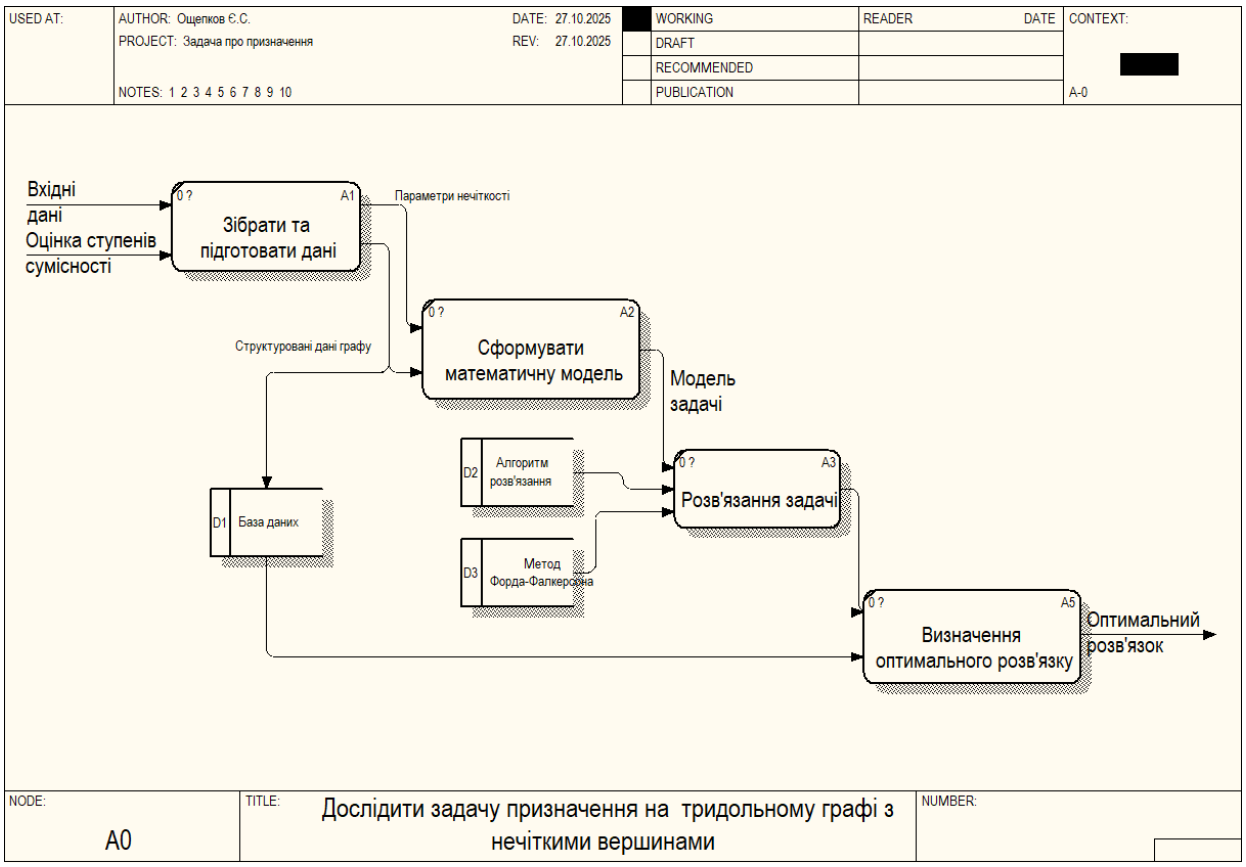

Рисунок 1.8 – DFD-діаграма 1-го рівня декомпозиції

1.2 Аналіз сценаріїв розв’язання задачі про призначення на тридольному графі з нечіткими вершинами

У процесі моделювання та дослідження задачі про призначення можливе використання різних підходів, що відрізняються способом подання невизначеності, типом математичного апарату та вимогами до вхідних даних. Розглянемо основні сценарії розв’язання цієї задачі, щоб обґрунтувати вибір найбільш відповідного методу для подальшого дослідження.

Сценарій 1 – класичний детермінований підхід.

Класичний підхід розв’язання задач про призначення ґрунтується на використанні чітко визначених числових даних. У цьому випадку тридольний граф $G = (V, E)$ містить вершини, ваги та ребра з відомими сталими значеннями. Задача зводиться до класичного пошуку поєднання елементів трьох множин V_1, V_2, V_3, яке максимізує сумарну вагу зв’язків. Такий сценарій має просту математичну модель та низьку обчислювальну складність. Але ігнорування природи нечіткості призводить до втрати важливої інформації про можливу варіативність параметрів. Рішення, отримане таким чином, може бути нестійким або неадекватним реальним умовам, де невизначеність є суттєвою.

Сценарій 2 – стохастичний підхід.

Цей сценарій моделює нечіткі характеристики як випадкові величини, для яких відомі або можуть бути оцінені ймовірнісні розподіли. Нечіткі ваги інтерпретуються як параметри цих розподілів (наприклад, математичні сподівання, дисперсії). Задача формулюється як задача стохастичного програмування, де ціллю може бути максимізація математичного сподівання сумарної ваги поєднання, або мінімізація ризику отримання низької ваги. Для розв'язання можуть використовуватися методи, такі як Sample Average Approximation (SAA) на основі симуляцій Монте-Карло, або складніші алгоритми стохастичного цілочисельного програмування. Такий підхід дозволяє враховувати невизначеність за допомогою імовірнісних моделей та надає інструменти для аналізу ризиків. Але він вимагає наявності даних для побудови або обґрунтування припущень про

конкретні ймовірнісні розподіли, що не завжди можливо або доцільно для нечіткої інформації, яка часто має експертне походження. Розв'язання задач стохастичного цілочисельного програмування є обчислювально дуже складним.

Сценарій 3 – підхід на основі теорії нечітких множин.

Цей підхід працює безпосередньо з нечіткими характеристиками, заданими функціями належності [4, 5, 6, 7]. Він відображає суб’єктивну, лінгвістичну невизначеність, коли параметри системи не можна виміряти точно, а лише оцінити експертно (наприклад, «висока сумісність», «помірна ефективність»). Для розв'язання пропонується ітераційний алгоритм, який зводить вихідну задачу до послідовності задач знаходження максимального потоку на допоміжних графах. Такий підхід природним чином моделює нечіткість та лінгвістичну невизначеність без необхідності переходу до імовірнісних моделей. Дозволяє знаходити компромісні рішення, що балансують між кількістю сформованих груп та їхньою мінімальною гарантованою надійністю.

Порівняємо кожен сценарій. Для порівняння сценаріїв використаємо метод аналізу ієрархій, який дозволяє оцінити кожен сценарій за рядом критеріїв. Це дозволяє визначити найбільш підходящий варіант для реалізації. Розглядатимемо наступні критерії оцінки:

– критерій 1: адекватність моделювання нечіткості (здатність сценарію коректно представляти та обробляти нечітку інформацію про сумісність);

– критерій 2: обчислювальна складність (ресурсоємність алгоритмів, необхідних для знаходження розв’язку);

– критерій 3: складність впровадження сценарію на практиці (складність реалізації);

– критерій 4: наскільки зрозумілим є отримане рішення для експерта (інтерпретованість результатів).

Можливі альтернативи:

– альтернатива 1: класичний підхід;

– альтернатива 2: стохастичний підхід;

– альтернатива 3: підхід на основі теорії нечітких множин.

Метод аналізу ієрархій, або метод Сааті, є підходом до прийняття рішень, що допомагає визначити пріоритети та відносну важливість різних факторів у складних системах.

Проводяться парні порівняння, де кожен елемент порівнюється з іншими щодо їх важливості відносно певного критерію. Для цього використовується шкала від 1 до 9, де 1 означає рівну важливість, а 9 абсолютну перевагу одного елемента над іншим. Результати цих порівнянь записуються у вигляді матриці парних порівнянь, з якої обчислюються локальні пріоритети елементів, що показують їх відносну важливість.

Використовуватимемо метод Сааті для ієрархічної структури, вид якої наведено на рисунку 1.9.

Формуємо матрицю попарних порівнянь для всіх елементів першого рівня ієрархії та обчислюємо вектор локальних пріоритетів (табл. 1.1).

Також обчислюємо індекс узгодженості, як суму елементів матриці за стовпцями.

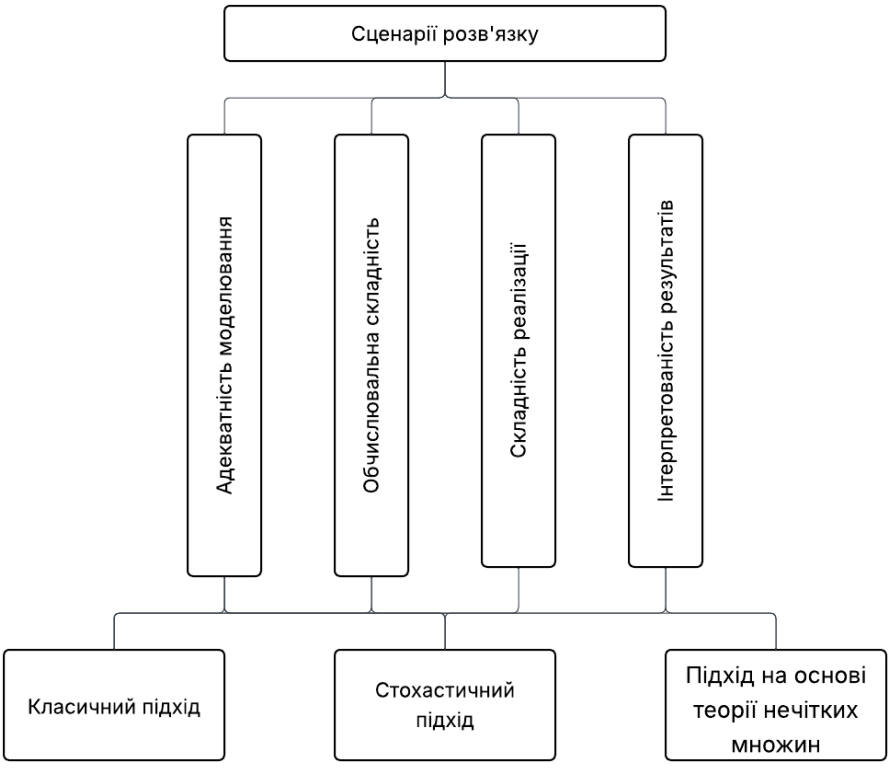

Рисунок 1.9 – Ієрархічна структура проблеми вибору

Таблиця 1.1 – Розрахунок вектору локальних пріоритетів критеріїв

	К1	К2	К3	К4	Середнє геометричне по рядках	Вектор пріоритетів
К1	1	4	6	8	$x_1 = \sqrt[4]{1 \cdot 4 \cdot 6 \cdot 8} = 3{,}7224$	$p_1^K = \frac{x_1}{\sum_{i=1}^{4} x_i} = 0{,}6305$
К2	$\frac{1}{4}$	1	2	4	$x_2 = \sqrt[4]{\frac{1}{4} \cdot 1 \cdot 2 \cdot 4} = 1{,}1892$	$p_2^K = \frac{x_2}{\sum_{i=1}^{4} x_i} = 0{,}2014$
К3	$\frac{1}{6}$	$\frac{1}{2}$	1	2	$x_3 = \sqrt[4]{\frac{1}{6} \cdot \frac{1}{2} \cdot 1 \cdot 2} = 0{,}6389$	$p_3^K = \frac{x_3}{\sum_{i=1}^{4} x_i} = 0{,}1082$
К4	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$	1	$x_4 = \sqrt[4]{\frac{1}{8} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot 1} = 0{,}3536$	$p_4^K = \frac{x_4}{\sum_{i=1}^{4} x_i} = 0{,}0599$
					$\sum_{i=1}^{4} x_i = 5{,}9041$	$\sum_{i=1}^{4} p_i^K = 1{,}0000$

Для обчислення індексу узгодженості знаходимо суми елементів матриці за стовпцями:

$$y_1 = 1{,}542,\ \ y_2 = 5{,}75,\ \ y_3 = 9{,}5,\ \ y_4 = 15.$$

Тоді:

$$\lambda_{\max}^K \approx \sum_{i=1}^{4} y_i p_i^k \approx 4{,}056.$$

Та індекс узгодженості:

$$CI^K = \frac{4{,}056 - 4}{4 - 1} = 0{,}019.$$

Оскільки матриця попарних порівнянь критеріїв є матрицею четвертого

порядку, то $RI^K = 0,90$, отже, відношення узгодженості дорівнює

$$CR^K = \frac{CI^K}{RI^K} = \frac{0,019}{0,9} = 0,021.$$

Оскільки відношення узгодженості дорівнює $0,021$, то будемо вважати, що матрицю попарних порівнянь критерію побудовано правильно.

Вектор локальних пріоритетів критеріїв відносно проблеми вибору має наступний вигляд:

$$\vec{p}^K = (0,6305; 0,2014; 0,1082; 0,0599)^T.$$

Ці значення показують відносну важливість кожного критерію у процесі прийняття рішень. Найвищий пріоритет має критерій з найбільшим значенням, що свідчить про його найбільшу вагу у системі оцінки.

Так само, як і для матриці попарних порівнянь критеріїв розрахуємо вектори локальних порівнянь для критеріїв.

Далі формуємо матриці попарних порівнянь альтернатив за кожним критерієм (табл. 1.2 – 1.5).

Таблиця 1.2 – Матриця попарних порівнянь альтернатив за К1

К1	А1	А2	А3
А1	1	$\frac{1}{3}$	$\frac{1}{7}$
А2	3	1	$\frac{1}{3}$
А3	7	3	1

Таблиця 1.3 – Матриця попарних порівнянь альтернатив за К2

К2	А1	А2	А3
А1	1	4	2
А2	$\frac{1}{4}$	1	$\frac{1}{3}$
А3	$\frac{1}{2}$	3	1

Таблиця 1.4 – Матриця попарних порівнянь альтернатив за К3

К3	А1	А2	А3
А1	1	5	3
А2	$\frac{1}{5}$	1	$\frac{1}{2}$
А3	$\frac{1}{3}$	2	1

Таблиця 1.5 – Матриця попарних порівнянь альтернатив за К4

К4	А1	А2	А3
А1	1	4	6
А2	$\frac{1}{4}$	1	3
А3	$\frac{1}{6}$	$\frac{1}{3}$	1

Для кожної матриці розраховуємо вектори локальних пріоритетів:

$$\vec{p}_1^{\,A} = \begin{pmatrix} 0,088 \\ 0,243 \\ 0,669 \end{pmatrix},\ \vec{p}_2^{\,A} = \begin{pmatrix} 0,558 \\ 0,122 \\ 0,320 \end{pmatrix},\ \vec{p}_3^{\,A} = \begin{pmatrix} 0,648 \\ 0,122 \\ 0,230 \end{pmatrix},\ \vec{p}_4^{\,A} = \begin{pmatrix} 0,691 \\ 0,218 \\ 0,091 \end{pmatrix}.$$

Так як, матриці попарних порівнянь третього порядку, то $RI^A = 0,58$. Обчислимо індекси та відношення узгодженостей для матриць парних порівнянь альтернатив за кожним із критеріїв.

$$CI_{K1}^A = 0,004,\ CI_{K2}^A = 0,009,\ CI_{K3}^A = 0,002,\ CI_{K4}^A = 0,027,$$
$$CR_{K1}^A = 0,006,\ CR_{K2}^A = 0,016,\ CR_{K3}^A = 0,003,\ CR_{K4}^A = 0,046.$$

Видно, що індекс узгодженості близький до 0,1. Це є прийнятним показником, який свідчить про високий ступінь узгодженості експертних суджень, що були використані для побудови матриць. Таким чином, можна вважати, що результат є достовірним та узгоджений з думками експерта.

Розрахуємо вектор глобальних пріоритетів альтернатив. Для цього з векторів локальних пріоритетів альтернатив за кожним критерієм складемо матрицю:

$$P^A = \begin{pmatrix} 0,088 & 0,558 & 0,648 & 0,691 \\ 0,243 & 0,122 & 0,122 & 0,218 \\ 0,669 & 0,320 & 0,230 & 0,091 \end{pmatrix},$$

$$\vec{p} = P^A \vec{p}^{\,K}.$$

Вектор глобальних пріоритетів дорівнює:

$$\vec{p} = \begin{pmatrix} 0{,}088 & 0{,}558 & 0{,}648 & 0{,}691 \\ 0{,}243 & 0{,}122 & 0{,}122 & 0{,}218 \\ 0{,}669 & 0{,}320 & 0{,}230 & 0{,}091 \end{pmatrix} \cdot \begin{pmatrix} 0{,}6305 \\ 0{,}2014 \\ 0{,}1082 \\ 0{,}0559 \end{pmatrix} = \begin{pmatrix} 0{,}280 \\ 0{,}204 \\ 0{,}516 \end{pmatrix}.$$

Розрахуємо індекс узгодженості та відношення узгодженості для всієї ієрархії:

$$CI = CI^K + (\vec{p}^K, \overrightarrow{CI^A}),$$

$$(\vec{p}^K, \overrightarrow{CI^A}) = \sum_{i=1}^{4} p_i^k CI_{Ki}^A = 0{,}0062,$$

$$CI = 0{,}019 + 0{,}0062 = 0{,}0252,$$

$$RI = RI^K + RI^A = 0{,}9 + 0{,}58 = 1{,}48,$$

$$CR = \frac{CI}{RI} = 0{,}017.$$

Остаточно отримане значення відношення узгодженості є дуже близьким до 0,1, і це є гарною узгодженістю.

Оскільки максимальна компонента вектора глобальних пріоритетів відповідає третій альтернативі (А3), то за сценарій розв’язання обираємо підхід на основі теорії нечітких множин.

1.3 Змістовна та формальна постановка задачі

Розглядаються задачі, в яких для трьох заданих множин об'єктів, що не перетинаються, потрібно знайти найбільш ефективний варіант формування максимального числа груп, що містять по одному об'єкту з кожної множини. Якість групи оцінюється ступенем істинності нечіткого висловлювання – «група задовольняє поставленим вимогам».

Задачі відрізняються тим, як визначається ця ступінь істинності. Наприклад в одній із задач вона дорівнює мінімальному зі ступенів істинності нечітких висловлювань «поєднання з першого і другого елементів задовольняє вимогам» і «поєднання з другого і третього елементів задовольняє вимогам». Скажімо, треба сформувати кілька груп чисельністю по три особи в кожній. При цьому перша особа вибирається з деякої множини студентів-магістрів, друга – лідер, вибирається з множини аспірантів, третя вибирається з множини студентів бакалаврату.

Відомий ступінь істинності нечіткого висловлювання «магістр спрацюється з лідером» і «лідер спрацюється з бакалавром» для кожної пари «лідер-магістр» і «лідер-бакалавр». Під ефективністю групи розуміємо ступінь істинності висловлювання «особовий склад групи спрацюється з лідером», під ефективністю варіанта формування необхідної кількості груп – ступінь істинності висловлювання «особовий склад кожної групи спрацюється зі своїм лідером».

Введемо поняття тридольного графа [8, 9] та розглядатимемо орієнтовані графи. Граф $G = (V, E)$ назвемо тридольним, якщо множина його вершин розпадається на три частини V_1, V_2, V_3, що не перетинаються. При цьому якщо $(u, v) \in E$, то або $u \in V_1$ та $v \in V_2$, або $u \in V_2$ та $v \in V_3$. Тридольний граф зображений на рисунку 1.10.

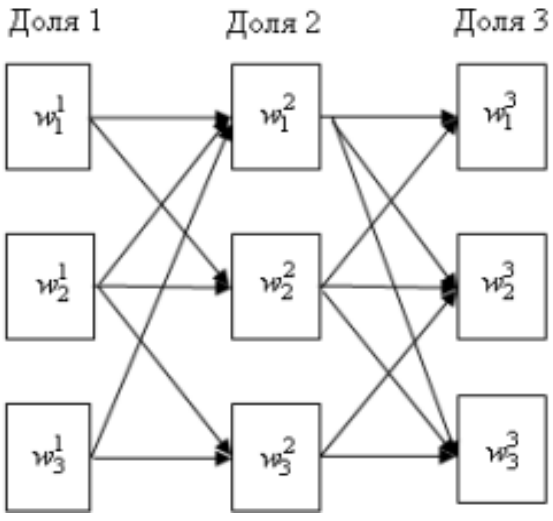

Рисунок 1.10 – Тридольний граф G

Тривершинним ансамблем назвемо шлях, що складається з двох дуг і містить три різні вершини по одній із кожної частки графа. Під вагою ансамблю розумітимемо мінімальну з ваг його ребер.

Тривершинним поєднанням P у графі $G=(V,E)$ назвемо таку множину тривершинних ансамблів з E, що будь-які два різних ансамбля з P не є суміжними, тобто не мають спільних вершин.

Потужністю тривершинного поєднання назвемо кількість тривершинних ансамблів у ньому. Під вагою тривершинного поєднання будемо розуміти найменшу з ваг його ансамблів. Тривершинне поєднання назвемо максимальним, якщо його потужність максимальна.

Отже, розглянемо орієнтований тридольний граф $G=(V,E)$. Множина вершин $V=V_1\cup V_2\cup V_3$, де V_1 – множина вершин першої долі графа, V_2 – множина вершин другої долі, V_3 – третьої. V_1, V_2, V_3 попарно не перетинаються. Нехай l_r – число вершин в V_r, $r=1,2,3$. Передбачається, що вершини кожної долі упорядковані.

Позначимо:

$$w_i^1(i=1,2,...,l_1) \text{ – вершина з } V_1 \text{ з номером } i,$$

$$w_j^2(j=1,2,...,l_2) \text{ – вершина з } V_2 \text{ з номером } j,$$

$$w_k^3(k=1,2,...,l_3) \text{ – вершина з } V_3 \text{ з номером } k.$$

Під вагою дуги розумітимемо надійність виконання відповідної вимоги [10–15] (санітар спрацюється з командиром, командир спрацюється з бійцем тощо). Потрібно побудувати тривершинне поєднання заданої потужності та найбільшої ваги. Під вагою поєднання розуміється мінімальна вага дуги з числа дуг, що входять до цього поєднання.

1.4 Постановка задач дослідження

Прикладом задачі дослідження може бути задача комплектування автономних рятувальних модулів для екстрених ситуацій. Необхідно сформувати рятувальні команди для дій у надзвичайних ситуаціях. Кожна команда складається з трьох елементів:

V_1 – спеціалісти-рятувальники, що мають різні рівні підготовки;

V_2 – автономні транспортні модулі, що можуть мати різні можливості;

V_3 – комплекти обладнання, необхідні для операцій.

Кожен рятувальник має нечітко визначений рівень підготовки, оскільки його ефективність залежить від умов роботи, стресу та ресурсів. Транспортні модулі змінюють характеристики залежно від прохідності, витрат пального та погодних умов. Комплекти обладнання оцінюються за нечіткими критеріями надійності й ефективності, бо їхня продуктивність у реальних умовах непередбачувана.

Отже, метою роботи є підвищення ефективності розподілу ресурсів у трикомпонентних системах шляхом дослідження математичної моделі, алгоритму та розробки програмного забезпечення для розв’язання задачі про призначення на тридольному графі з нечіткими вершинами. Для досягнення поставленої мети необхідно виконати наступні завдання:

– провести огляд і аналіз сучасного стану задачі «про призначення на тридольному графі з нечіткими вершинами»;

– обрати метод дослідження поставленої задачі та дослідити алгоритм її розв’язання;

– самостійно попрацювати з літературою, провести дослідження, підібрати та опрацювати матеріали, що включає пошук та вивчення наукових статей, книг та інших джерел, які стосуються теми дослідження;

– застосовувати методи та інструменти системного аналізу для результативного та ефективного дослідження систем та процесів різної природи;

– виконати програмну реалізацію мовою Python дослідженого методу у вигляді додатку з графічним інтерфейсом;

– провести обчислювальний експеримент для перевірки коректності та ефективності роботи програми на прикладі задачі комплектування автономних рятувальних груп.

2 ВИБІР ТА ОБҐРУНТУВАННЯ МЕТОДУ РОЗВ'ЯЗАННЯ

2.1 Побудова допоміжного графа для задачі про призначення

Для розв'язку задачі для заданого графа G будується граф G^* [16, 17, 18]. Граф G^* зображено на рисунку 2.1.

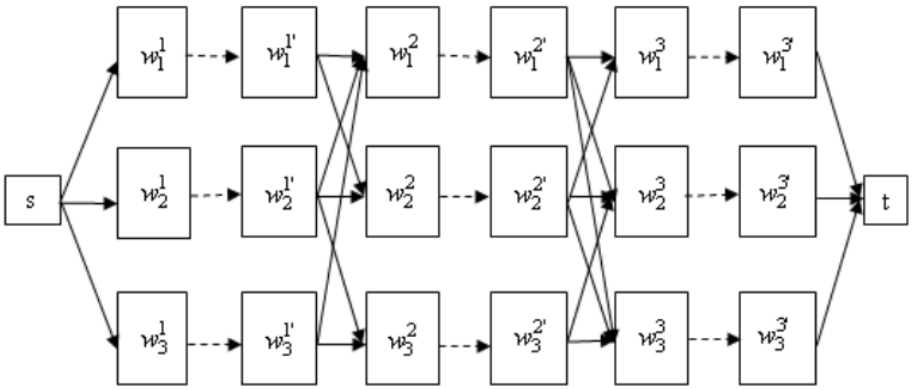

Рисунок 2.1 – Граф G^*

Опишемо алгоритм побудови графа G^*.

Кожну вершину u графа G замінимо на дугу (u,u'). Якщо на графі G є дуга (u,v), то на графі G^* є дуги (u,u') та (u',v). Крім цього, на графі G^* є вершина s – джерело, з'єднана дугами з усіма вершинами першої долі графа G, і вершина t – стік, з'єднана дугами з усіма вершинами третьої долі графа G.

Розглядаючи граф G^* як мережу з пропускними здатностями дуг, що дорівнюють одиниці, знайдемо максимальний потік на цій мережі з джерела у стік.

Дугу, обсяг перевезення по якій більше нуля (а отже, дорівнює одиниці), назвемо завантаженою дугою, шлях від джерела до стоку, усі дуги якого є завантаженими дугами, назвемо завантаженим шляхом.

Розглянемо на графі G тривершинний ансамбль (w_i^1, w_j^2), (w_j^2, w_k^3). На графі G^* є єдиний шлях, що містить ці вершини – це шлях, що проходить через

вершини s, w_i^1, $w_i^{1'}$, w_j^2, $w_j^{2'}$, w_k^3, $w_k^{3'}$, t. Надалі вважатимемо, що цей шлях відповідає ансамблю, що розглядається, і навпаки. Отже, кожному тривершинному ансамблю на G відповідає певний шлях на графі G^* і навпаки.

Завантажені шляхи на графі G^* не перетинаються. Дійсно, в силу умов, що накладаються на потік, обсяг підвезення в кожен пункт повинен дорівнювати обсягу вивезення з нього. На графі G^* у кожну вершину, не враховуючи джерела і стоку, входить або виходить точно одна дуга. Отже, жодна із зазначених вершин не може лежати на двох завантажених шляхах.

Зауваження 2.1. Число завантажених шляхів дорівнює величині максимального потоку.

Зауваження 2.2. Оскільки завантаженим шляхам, що не перетинаються, відповідають тривершинні ансамблі, що не перетинаються, то величина максимального потоку не перевищує потужності тривершинного поєднання.

Непересічним тривершинним ансамблям відповідають непересічні шляхи від джерела до стоку на графі G^*. Це випливає безпосередньо з визначення відповідності шляхів і ансамблів. Отже, максимальне число непересічних тривершинних ансамблів у графі G (потужність максимального тривершинного поєднання) не перевищує величини максимального потоку.

Із зауважень 2.1 і 2.2 випливає, що величина максимального потоку на G^* дорівнює потужності максимального тривершинного поєднання на G.

2.2 Алгоритм розв'язання задачі про призначення

Для розв'язання задачі з відшукання максимального тривершинного поєднання найбільшої ваги на графі G пропонується наступний алгоритм, що складається з послідовності етапів.

На підготовчому етапі за графом G будується граф G^*, який ми позначимо G_1^*. На етапі з номером τ розглядаємо граф G_τ^*. Для нього знаходимо мак-

симальний потік (його величину v_τ та завантажені шляхи), наприклад, методом Форда-Фалкерсона [19]. За знайденими завантаженими шляхами знаходимо відповідне їм тривершинне поєднання. Визначаємо його вагу, нехай вона дорівнює g_τ. Прибираємо з графа G_1^* дуги, вага яких не перевищує g_τ. Якщо при цьому з'являється ізольована вершина, то прибираємо і її. Отримуємо граф $G_{\tau+1}^*$. Переходимо до наступного етапу.

Обчислення проводимо доти, доки не отримаємо максимальний потік (тривершинне поєднання), величина якого менша за v_1 (менша за отриману на першому етапі). Тривершинне поєднання, отримане на передостанньому етапі, є шуканим.

Запропонований підхід узагальнюється на випадок багатодольного графа з кінцевим числом долей.

2.3 Алгоритм Форда-Фалкерсона

Нехай задано множину вершин V, у якої виділено дві вершини: s (витік, вхід або джерело) і t (вихід або стік), інші вершини називатимемо проміжними вузлами.

Нехай визначена функція $c: V \times V \to R$, яка задовольняє співвідношенням

$$c(x, y) \geq 0,$$
$$c(x, s) = 0,$$
$$c(t, y) = 0$$

для будь-яких вершин $x, y \in V$. Тоді $G = (V, s, t, c)$ – мережа, функція $c(s,t)$ – пропускна спроможність мережі G, а функція $c(x, y)$ – пропускна спроможність ребра (x, y).

Нехай G – мережа, а функція $f : V \times V \to R$ задовольняє наступним умовам:

$$\forall x, y \in V \quad f(x,y) \le c(x,y),$$
$$\forall x, y \in V \quad f(x,y) = -f(y,x),$$
$$\forall v \in V, v \ne s,t \quad \sum_{x \in V} f(v,x) = 0.$$

Тоді f – потік в мережі G. Число $|f| = \sum_{x \in V} f(s,x)$ називають величиною потоку. Потік в мережі G з максимальною величиною називається максимальним.

Нехай G – мережа, а множина її вершин V розбита на дві множини, що не перетинаються, $s \in S$, $t \in T$. Тоді (S,T) – розріз мережі G. Величина $c(S,T) = \sum_{x \in S, y \in T} c(x,y)$ називається пропускною спроможністю розрізу. Будь-який розріз мережі G з мінімальною пропускною спроможністю називається мінімальним.

Теорема. У цілочисельній мережі G (у такій, що має цілочисельні пропускні спроможності) існує максимальний потік, при чому серед максимальних потоків даної мережі знайдеться цілочисельний.

Теорема Форда-Фалкерсона. У мережі $G = (V,s,t,c)$ заданий потік f. Тоді три наступні твердження є рівносильними:

– потік f максимальний;

– існує такий розріз (S,T), що $|f| = c(S,T)$;

– у залишковій мережі G_f немає доповнюючого шляху.

Іншими словами, величина максимального потоку із джерела s у стік t дорівнює пропускній спроможності мінімального розрізу, що відокремлює ці дві вершини.

Алгоритм або метод Форда-Фалкерсона – це жадібний алгоритм, що обчислює максимальний потік у мережі.

Ідея алгоритму полягає в наступному. Ми вибираємо такий шлях від джерела до стоку, щоб для кожного ребра залишкова пропускна здатність була строго більше нуля. При цьому ребра на даному шляху можуть проходитися як у прямому, так і в зворотному напрямку. Вибираємо мінімальне значення серед залишкових пропускних спроможностей ребер даного шляху. Збільшуємо потік на кожному з ребер даного шляху на обране мінімальне значення. Далі шукаємо наступний аналогічний шлях. Робота алгоритму продовжується до тих пір, поки вдається знаходити дані шляхи. Відразу відзначимо, що даний алгоритм відноситься до класу недетермінованих, тобто кожен наступний крок алгоритму визначено неоднозначно. І час роботи (кількість кроків) алгоритму залежить від того, як будуть вибиратися кроки.

Алгоритм Форда-Фалкерсона.

Крок 1. Вважаємо, що початковий потік мережею є нульовим: $\forall e(x,y) \in E \quad f(x,y) = 0$.

Крок 2. У залишковій мережі шукаємо будь-який шлях з джерела до стоку, дуги якого задовольняють умову: $f(x,y) \le c(x,y)$. У разі, якщо такого потоку не існує, то наявний у мережі потік і є максимальним.

Крок 3. Через знайдений шлях (будемо називати збільшувальним шляхом) пускаємо максимальний можливий потік.

Крок 4. На знайденому збільшуваному шляху в залишковій мережі шукаємо ребро з мінімальною пропускною здатністю $c_{\min}$.

Крок 5. Для кожного ребра на знайденому збільшувальному шляху збільшуємо потік на $c_{\min}$, а на протилежному – зменшуємо на $c_{\min}$.

Крок 6. Модифікуємо залишкову мережу: для усіх ребер на знайденому збільшувальному шляху, а також для протилежних їм ребер, обчислюємо нову пропускну спроможність. Якщо нова пропускна спроможність не дорівнює нулю, додаємо ребро до залишкової мережі, а якщо дорівнює нулю, стираємо його.

Крок 7. Повертаємося на крок 2.

2.4 Алгоритм Едмондса-Карпа

Метод Форда-Фалкерсона, описаний в пункті 2.3, визначає загальну схему знаходження максимального потоку, але не конкретизує спосіб вибору збільшувального шляху на кожній ітерації. Невдалий вибір шляхів може призвести до значної кількості ітерацій і, як наслідок, до низької ефективності роботи алгоритму. Також у випадку ірраціональних пропускних здатностей алгоритм може навіть не збігатися.

Для усунення цієї невизначеності використовується алгоритм Едмондса-Карпа. Це конкретизація методу Форда-Фалкерсона, в якій для знаходження збільшувального шляху в залишковій мережі використовується пошук у ширину (BFS) [20, 21].

Основна ідея полягає в тому, що на кожному кроці обирається найкоротший збільшувальний шлях із джерела s до стоку t. Під найкоротшим шляхом розуміється такий шлях, що містить мінімальну кількість дуг, незалежно від їх пропускної здатності.

Використання пошуку в ширину гарантує, що довжина найкоротшого шляху від джерела до будь-якої вершини в залишковій мережі не зменшується з кожною ітерацією. Ця властивість забезпечує завершення алгоритму за скінченну кількість кроків.

Формально алгоритм можна описати наступним чином.

Нехай задано транспортну мережу $G = (V, E)$ з джерелом s, стоком t та функцією пропускної здатності $c(u,v) > 0$. Позначимо:

– $f(u,v)$ – потік по дузі;

– $c_f(u,v)$ – залишкова пропускна здатність дуги, яка визначається як:

$$c_f(u,v) = c(u,v) - f(u,v);$$

– G_f – залишкова мережа, що складається з дуг, для яких $c_f(u,v) > 0$.

Алгоритм виконується за наступною схемою.

Крок 1. Встановити потік $f(u,v)=0$ для всіх дуг $(u,v)\in E$.

Крок 2. Використовуючи пошук в ширину, знайти найкоротший шлях P від s до t у залишковій мережі G_f (шлях складається з дуг, де $c_f(u,v)>0$).

Крок 3. Якщо такого шляху не існує, алгоритм завершує роботу і поточний потік f є максимальним.

Крок 4. Знайти резервну пропускну здатність шляху P:

$$\Delta = \min_{(u,v)\in P}\{c_f(u,v)\}.$$

Крок 5. Для кожної дуги (u,v), що входить до знайденого шляху P збільшити потік у прямому напрямку $f(u,v)=f(u,v)+\Delta$ та зменшити потік у зворотному напрямку $f(v,u)=f(v,u)-\Delta$ (це еквівалентно створенню зворотного ребра в залишковій мережі, що дозволяє скасовувати потік на наступних ітераціях).

Крок 6. Перейти до кроку 2.

Обчислювальна складність алгоритму Едмондса-Карпа у загальному випадку складає $O(V\cdot E^2)$, де V – кількість вершин, а E – кількість дуг у графі.

Для задачі, що розглядається в даній роботі, цей алгоритм є особливо ефективним, тому що побудований допоміжний граф G^* є мережею з одиничними пропускними здатностями. У таких мережах складність алгоритму Едмондса-Карпа значно менша і становить $O(\min(V^{\frac{2}{3}},E^{\frac{1}{2}})\cdot E)$.

Оскільки всі пропускні здатності є цілими числами, алгоритм гарантовано знаходить точний розв'язок – цілочисельний потік, що відповідає дискретній природі задачі про формування груп.

Також використання стандартного пошуку в ширину дозволяє легко програмно реалізувати алгоритм за допомогою базових структур даних.

Висновки за розділом 2

У другому розділі розглянуто обґрунтування вибору алгоритмічних методів для розв’язання задачі про призначення на тридольному графі з нечіткими вершинами.

Детально описано процедуру побудови допоміжного графа (транспортної мережі), що включає введення фіктивних вершин джерела та стоку, а також застосування методу розщеплення вершин. Такий підхід дозволяє звести задачу комбінаторної оптимізації до класичної задачі про максимальний потік, гарантуючи виконання умови, що кожен об'єкт системи використовується не більше одного разу.

Досліджено ітераційний алгоритм пошуку оптимального рішення, який поєднує методи теорії графів з логікою прийняття рішень в умовах невизначеності. Алгоритм дозволяє знаходити оптимальні варіанти розподілу ресурсів, максимізуючи кількість сформованих груп при забезпеченні найвищого можливого рівня надійності найслабшої ланки.

Розглянуто теоретичні основи методу Форда-Фалкерсона. Для практичної реалізації обґрунтовано вибір алгоритму Едмондса-Карпа. Показано, що використання пошуку в ширину (BFS) для знаходження найкоротших збільшувальних шляхів є ефективним рішенням для мереж з одиничними пропускними здатностями, забезпечуючи поліноміальну обчислювальну складність та гарантовану збіжність.

Отже, обрана комбінація методів, таких як побудова допоміжної мережі, алгоритм Едмондса-Карпа та ітеративна фільтрація за нечіткими вагами, створює теоретичну основу для подальшої програмної реалізації системи підтримки прийняття рішень.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Мова програмування Python

Мова програмування Python [22, 23, 24] є сучасним високорівневим інструментом, який широко застосовується у наукових обчисленнях, інженерії даних, моделюванні алгоритмів та для створення прототипів складних програмних систем. З моменту появи у 1991 році Python поступово став однією з найбільш популярних мов завдяки своїй простоті у використанні та підтримці різних парадигм програмування.

Мова Python підтримує процедурне, об’єктно-орієнтоване та функціональне програмування, що робить її універсальним інструментом для розв’язання різних задач – від базових обчислень до аналізу великих даних та побудови графів. Важливою її перевагою є динамічна типізація та автоматичне керування пам'яттю, що значно спрощує процес розробки та дозволяє програмісту зосередитися на вирішенні прикладної задачі, а не на технічних деталях реалізації.

Однією з особливостей Python є лаконічний синтаксис, що орієнтований на легке читання і швидке написання коду. Завдяки високому рівню абстракції, Python дозволяє сфокусуватися на логіці алгоритму, а не на низькорівневих деталях керування пам'яттю, що значно скорочує час реалізації програмного продукту. Саме тому Python було обрано як інструмент для побудови алгоритму розв’язку задачі призначення на тридольному графі з нечіткими вершинами.

Python є стандартом у сфері наукових досліджень, особливо пов'язаних з теорією графів та мережевим аналізом. Наявність спеціалізованих бібліотек дозволяє ефективно моделювати складні структури даних, такі як тридольні графи. Зокрема, у ході виконання роботи було використано ряд спеціалізованих бібліотек та модулів [25, 26]:

– Tkinter – стандартна бібліотека для створення графічного інтерфейсу користувача (GUI) [27]. Вона входить до базового дистрибутива Python, що спрощує розгортання програми. У роботі Tkinter використовується для ство-

рення вікон, полів введення параметрів, кнопок керування процесом та відображення логів розв'язання;

– NetworkX – бібліотека для створення та маніпулювання структурою складних мереж [28]. У даній роботі NetworkX використовується для візуалізації тридольного графа на кожному етапі алгоритму. Хоча основна логіка пошуку максимального потоку, а саме метод Едмондса-Карпа, реалізована власноруч (для забезпечення гнучкості алгоритму), NetworkX надає зручні інструменти для відображення графової структури;

– Matplotlib – бібліотека для візуалізації даних [29, 30]. Вона інтегрується з Tkinter і дозволяє виводити побудовані графи безпосередньо у вікно програми, забезпечуючи наочність процесу відсіювання ненадійних зв'язків.

Ще однією особливістю мови є її інтерпретованість, тобто відсутність потреби в компіляції. Це дає змогу тестувати окремі частини алгоритму, виконувати покрокову відладку та інтерактивно перевіряти математичні розрахунки. Такий підхід підвищує ефективність дослідження, оскільки дає змогу швидко виявляти помилки моделі. В задачах, що потребують частої зміни параметрів, така гнучкість є дуже корисною.

Також програмне забезпечення, розроблене на Python, може функціонувати на різних операційних системах без необхідності внесення змін у вихідний код.

Разом із перевагами Python має і певні недоліки. Його продуктивність нижча порівняно з компільованими мовами, такими як C++ або Java, але у задачах середньої розмірності, подібних до досліджуваної, цей недолік практично не впливає на швидкість виконання алгоритмів. Також Python має особливість, що полягає в наявності глобального блокування інтерпретатора, що обмежує можливість паралельного виконання потоків. Проте в даній роботі алгоритми виконуються послідовно, тому ця особливість не створює обмежень.

Незважаючи на те, що Python є інтерпретованою мовою і може поступатися у швидкості виконання компільованим мовам, для розв'язання задач комбінаторної оптимізації на графах тієї розмірності, що розглядається в роботі,

його продуктивності цілком достатньо.

Таким чином, використання Python та описаних вище бібліотек є оптимальним вибором, що дозволяє поєднати надійну математичну логіку зі зручним інтерфейсом та якісною візуалізацією результатів.

3.2 Алгоритм розв’язання задачі про призначення на тридольному графі з нечіткими вершинами

Програмна реалізація базується на модифікованому підході до знаходження максимального тривершинного поєднання на графі. Процес розв’язання задачі можна розділити на кілька етапів, які циклічно повторюються для покращення якості розв’язку.

Етап 1. Побудова допоміжної транспортної мережі G^*.

Оскільки вихідна задача передбачає, що кожен об'єкт (спеціаліст, модуль, обладнання) може бути використаний лише один раз, алгоритм виконує трансформацію вихідного тридольного графа у транспортну мережу з одиничними пропускними здатностями.

Процедура побудови мережі виконує наступні кроки:

– створюються фіктивні вершини: джерело s та стік t;

– для виконання умови унікальності використання ресурсів кожна вершина u вихідного графа розбивається на дві: вхідну u та вихідну u'. Між ними додається ребро (u,u') з пропускною здатністю 1;

– до графу додаються дуги між компонентами системи лише за умови, що вага дуги (надійність) перевищує поточний поріг відсікання;

– джерело s з'єднується з усіма вершинами першої долі, а всі вершини третьої долі з'єднується зі стоком t.

Етап 2. Знаходження максимального потоку (метод Едмондса-Карпа).

Для визначення максимально можливої кількості груп (потужності поєднання) застосовується алгоритм Едмондса-Карпа. Це реалізація методу Форда-

Фалкерсона, яка використовує пошук у ширину (BFS) для знаходження найкоротших доповнювальних шляхів у залишковій мережі.

Логіка роботи алгоритму:

– за допомогою BFS у мережі шукається шлях від джерела до стоку, всі ребра якого мають вільну пропускну здатність;

– якщо шлях знайдено, по ньому пропускається потік величиною 1 (оскільки мережа є одиничною);

– оновлюється залишкова мережа: для кожного ребра прямого шляху пропускна здатність зменшується, а для зворотного ребра – збільшується;

– процес повторюється доти, доки неможливо знайти жодного шляху від джерела до стоку.

Етап 3. Ітеративна оптимізація надійності.

Алгоритм працює ітеративно, поступово підвищуючи вимоги до якості сформованих груп. Основний цикл працює за наступною схемою:

– на першому кроці поріг надійності встановлюється на мінімальне технічне значення (-1), щоб врахувати всі можливі зв'язки;

– будується мережа та знаходиться максимальний потік v (кількість груп);

– якщо поточний потік v менший за максимальний потік, знайдений на попередніх етапах, це означає, що підвищення вимог до надійності призвело до втрати кількості сформованих груп і алгоритм зупиняється, а оптимальним вважається рішення, отримане на попередньому кроці;

– серед усіх сформованих груп знаходиться та, що має найменшу надійність і це значення стає новим порогом відсікання;

– усі зв'язки, вага яких не перевищує знайдений мінімум, вилучаються з розгляду, і процес повторюється.

Розв’язок забезпечує максимально можливу кількість призначень при максимально можливому рівні надійності для найслабшої ланки системи.

3.3 Опис програми

3.3.1 Графічний інтерфейс користувача

Розроблена програма являє собою додаток, що реалізує алгоритм, описаний у розділі 3.2. Застосунок складається з двох логічних модулів: модуля обчислювального ядра (клас FuzzyAssignmentSolver), що відповідає за реалізацію алгоритму на графах, та модуля графічного інтерфейсу (клас App), що забезпечує взаємодію з користувачем та візуалізацію.

Графічний інтерфейс (рис. 3.1) розроблено з використанням бібліотеки Tkinter. Головне вікно програми створено таким чином, щоб забезпечити інтуїтивно зрозумілий процес введення даних та аналізу результатів. Вікно розділене на функціональні зони.

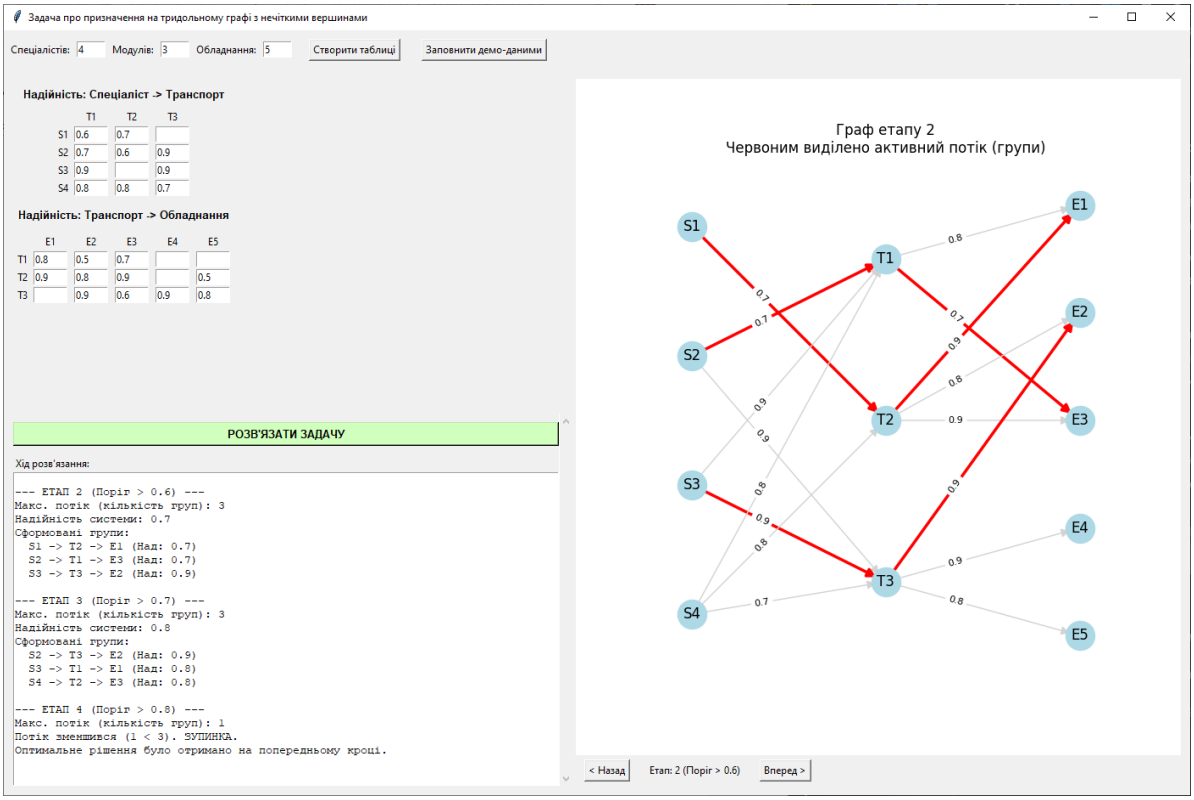

Рисунок 3.1 – Графічний інтерфейс

Панель конфігурації знаходиться у верхній частині вікна. Дозволяє задати

розмірність задачі (кількість вершин у кожній з трьох доль графа: V_1, V_2, V_3). Тут розташовані кнопки для генерації таблиць введення та завантаження тестових даних.

Область введення даних розташована у лівій частині. Містить динамічно згенеровані таблиці для введення нечітких оцінок надійності зв'язків між об'єктами.

Область керування та логування знаходиться під таблицями. Містить кнопку запуску розрахунку «Розв’язати задачу» та текстове поле, куди виводиться детальний звіт роботи алгоритму (покрокова зміна максимального потоку, поточний поріг надійності, склад сформованих груп).

Область візуалізації займає праву частину вікна. Тут за допомогою інтеграції Matplotlib та NetworkX відображається граф на поточному етапі розв'язання. Реалізовано кольорове кодування: активні зв'язки, що увійшли до оптимального рішення, підсвічуються червоним кольором.

Панель навігації дозволяє перемикатися між ітераціями алгоритму («Назад», «Вперед») для візуального аналізу динаміки відсіювання ненадійних зв'язків.

3.3.2 Структура та логіка програмного коду

Розглянемо детальну реалізацію основних компонентів програми.

Клас FuzzyAssignmentSolver (обчислювальне ядро) інкапсулює всю логіку розв'язання задачі. При ініціалізації він приймає списки імен об'єктів та словники з вагами зв'язків.

Побудова транспортної мережі. Метод build_network (рис. 3.2) відповідає за трансформацію вихідних даних у граф G^*. На цьому етапі відбувається фільтрація ребер. Якщо вага зв'язку менша або дорівнює поточному порогу (min_reliability_threshold), ребро не додається до графа.

Важливою особливістю реалізації є розщеплення вершин. Для кожного

об'єкта (окрім джерела та стоку) створюються дві вершини (v_in, v_out) та ребро між ними з пропускною здатністю 1. Це гарантує, що кожен об'єкт буде використаний у розподілі не більше одного разу.

```
def build_network(self, min_reliability_threshold):
    """Будує транспортну мережу G*, фільтруючи ребра за вагою"""
    self.graph = {}
    self.capacity = {}
    self.flow = {}

    # 1. Джерело -> Спеціалісти
    for s in self.specialists:
        s_in, s_out = f"{s}_in", f"{s}_out"
        self.add_edge(self.SOURCE, s_in, 1)
        self.add_edge(s_in, s_out, 1) # Розщеплення вершини

        # 2. Спеціаліст -> Транспорт
        for t in self.transports:
            weight = self.connections_s_t.get((s, t), 0)
            if weight > min_reliability_threshold:
                t_in = f"{t}_in"
                self.add_edge(s_out, t_in, 1)

    # 3. Транспорт -> Обладнання
    for t in self.transports:
        t_in, t_out = f"{t}_in", f"{t}_out"
        self.add_edge(t_in, t_out, 1) # Розщеплення вершини

        for e in self.equipments:
            weight = self.connections_t_e.get((t, e), 0)
            if weight > min_reliability_threshold:
                e_in = f"{e}_in"
                self.add_edge(t_out, e_in, 1)

    # 4. Обладнання -> Стік
    for e in self.equipments:
        e_in, e_out = f"{e}_in", f"{e}_out"
        self.add_edge(e_in, e_out, 1) # Розщеплення вершини
        self.add_edge(e_out, self.SINK, 1)
```

Рисунок 3.2 – Метод побудови транспортної мережі

Пошук максимального потоку. Для знаходження максимального потоку реалізовано алгоритм Едмондса-Карпа. Метод bfs (рис.3.3) – пошук у ширину, що знаходить найкоротший шлях від джерела до стоку в залишковій мережі. Метод edmonds_karp (рис. 3.4) використовує знайдені шляхи для збільшення

потоку. Важливим моментом є оновлення залишкової мережі: при проходженні потоку по ребру (u,v) його пропускна здатність зменшується, а пропускна здатність зворотного ребра (v,u) збільшується, що дозволяє алгоритму скасовувати раніше прийняті рішення для знаходження оптимуму.

```python
def bfs(self, parent):
    """Пошук у ширину (BFS) для алгоритму Едмондса-Карпа"""
    visited = {self.SOURCE}
    queue = collections.deque([self.SOURCE])
    while queue:
        u = queue.popleft()
        if u == self.SINK: return True
        for v in self.graph.get(u, []):
            # Якщо не відвідано І є залишкова пропускна здатність
            if v not in visited and self.capacity.get((u, v), 0) - self.flow.get((u, v), 0) > 0:
                visited.add(v)
                parent[v] = u
                queue.append(v)
    return False
```

Рисунок 3.3 – Пошук в ширину

```python
def edmonds_karp(self):
    """Знаходить максимальний потік у мережі"""
    self.flow = {k: 0 for k in self.capacity}
    max_flow = 0
    parent = {}

    while self.bfs(parent):
        path_flow = float('Inf')
        s = self.SINK

        # Знаходимо вузьке місце в знайденому шляху
        while s != self.SOURCE:
            path_flow = min(path_flow, self.capacity[(parent[s], s)] - self.flow[(parent[s], s)])
            s = parent[s]

        max_flow += path_flow

        # Оновлюємо потоки (прямий і зворотний)
        v = self.SINK
        while v != self.SOURCE:
            u = parent[v]
            self.flow[(u, v)] += path_flow
            self.flow[(v, u)] -= path_flow
            v = u

    return max_flow
```

Рисунок 3.4 – Метод Едмондса-Карпа

Інтерпретація результатів. Метод get_groups (рис. 3.5) аналізує отриманий потік. Він сканує граф і відновлює ланцюжки $S \to T \to E$, через які проходить одиничний потік. Для кожної знайденої групи обчислюється її надійність як мінімум з ваг ребер, що до неї входять. Також метод визначає загальну надійність системи на поточній ітерації.

```
def get_groups(self):
    """Витягує сформовані групи з поточного потоку"""
    groups = []
    min_sys_rel = 1.0

    # Проходимо по потоку, щоб відновити зв'язки
    for s in self.specialists:
        s_out = f"{s}_out"
        if s_out in self.graph:
            for neighbor in self.graph[s_out]:
                # Якщо є потік S -> T
                if self.flow.get((s_out, neighbor), 0) == 1:
                    t_node_in = neighbor
                    t_name = t_node_in.replace("_in", "")
                    t_out = f"{t_name}_out"

                    # Якщо є потік T -> E
                    e_name = None
                    if t_out in self.graph:
                        for e_neighbor in self.graph.get(t_out, []):
                            if self.flow.get((t_out, e_neighbor), 0) == 1:
                                e_name = e_neighbor.replace("_in", "")
                                break

                    if t_name and e_name:
                        w1 = self.connections_s_t.get((s, t_name), 0)
                        w2 = self.connections_t_e.get((t_name, e_name), 0)
                        g_rel = min(w1, w2)
                        min_sys_rel = min(min_sys_rel, g_rel)
                        groups.append((s, t_name, e_name, g_rel))

    # Якщо груп немає, повертаємо 0 надійності
    return groups, min_sys_rel if groups else 0
```

Рисунок 3.5 – Метод get_groups

Основний цикл розв'язання. Метод solve, зображений на рисунках 3.6 та 3.7, реалізує ітераційну процедуру покращення розв'язку. Алгоритм починає з нульового порогу відсікання. На кожній ітерації:

– будується мережа з урахуванням поточного порогу;

– знаходиться максимальний потік (кількість груп);

– якщо потік зменшився порівняно з максимумом – цикл переривається, оптимальним вважається попереднє рішення;

– якщо потік зберігся – поріг підвищується до рівня найслабшої ланки в поточних групах. Всі проміжні стани зберігаються у списку history для подальшої візуалізації.

```python
def solve(self):
    """Основний цикл розв'язання задачі"""
    self.history = []
    current_threshold = -1.0
    iteration = 1
    max_groups = -1

    logs = []

    while True:
        step_log = f"--- ЕТАП {iteration} (Поріг > {current_threshold}) ---\n"

        self.build_network(current_threshold)
        flow_val = self.edmonds_karp()
        groups, reliability = self.get_groups()

        step_log += f"Макс. потік (кількість груп): {flow_val}\n"

        # Зберігаємо стан для візуалізації
        snapshot = {
            'iteration': iteration,
            'threshold': current_threshold,
            'flow_val': flow_val,
            'groups': groups,
            'active_edges': [],
            'all_edges': []
        }

        # Збираємо дані для візуалізації (які ребра доступні, які активні)
        # Малюємо "Логічний граф" (S->T->E), а не повну транспортну мережу
        for s in self.specialists:
            for t in self.transports:
                w = self.connections_s_t.get((s, t), 0)
                if w > current_threshold:
                    is_active = False
                    # Перевіряємо, чи йде потік через це ребро
                    if self.flow.get((f"{s}_out", f"{t}_in"), 0) == 1:
                        is_active = True
                    snapshot['all_edges'].append(((s, t), w, is_active))
```

Рисунок 3.6 – Основний цикл розв’язку (частина 1)

```
        for t in self.transports:
            for e in self.equipments:
                w = self.connections_t_e.get((t, e), 0)
                if w > current_threshold:
                    is_active = False
                    if self.flow.get((f"{t}_out", f"{e}_in"), 0) == 1:
                        is_active = True
                    snapshot['all_edges'].append(((t, e), w, is_active))

        self.history.append(snapshot)

        if flow_val == 0:
            step_log += "Неможливо сформувати жодної групи.\n"
            logs.append(step_log)
            break

        if max_groups == -1:
            max_groups = flow_val
        elif flow_val < max_groups:
            step_log += f"Потік зменшився ({flow_val} < {max_groups}). ЗУПИНКА.\n"
            step_log += "Оптимальне рішення було отримано на попередньому кроці.\n"
            logs.append(step_log)
            break

        step_log += f"Надійність системи: {reliability}\n"
        step_log += "Сформовані групи:\n"
        for g in groups:
            step_log += f"  {g[0]} -> {g[1]} -> {g[2]} (Над: {g[3]})\n"

        logs.append(step_log)
        # Наступний поріг відсікання дорівнює поточній мінімальній надійності
        current_threshold = reliability
        iteration += 1

    return logs
```

Рисунок 3.7 – Основний цикл розв’язку (частина 2)

Клас App (графічний інтерфейс та візуалізація) відповідає за взаємодію з користувачем. Метод run_solver (рис. 3.8) збирає дані з полів введення, валідує їх (перевіряє на коректність чисел) та передає у FuzzyAssignmentSolver.

Метод update_plot (рис 3.9) відповідає за візуалізацію графа та розташовує об'єкти пошарово (Спеціалісти – ліворуч, Транспорт – посередині, Обладнання – праворуч) та вирівнює їх по вертикалі. За допомогою бібліотеки NetworkX будується граф, де ребра, що входять до активного потоку, забарвлюються у червоний колір та мають більшу товщину, а неактивні (але доступні) ребра відображаються сірим. Це дозволяє користувачу наочно оцінити структуру розв'язку.

```python
def run_solver(self):
    # 1. Збір даних з таблиць
    conn_st = {}
    conn_te = {}

    try:
        for (s, t), entry in self.entries_st.items():
            val = entry.get().strip()
            if val and val != '-':
                conn_st[(s, t)] = float(val)

        for (t, e), entry in self.entries_te.items():
            val = entry.get().strip()
            if val and val != '-':
                conn_te[(t, e)] = float(val)
    except ValueError:
        messagebox.showerror("Помилка", "Перевірте правильність введених чисел (використовуйте крапку для дробів)")
        return

    # 2. Запуск алгоритму
    self.solver_instance = FuzzyAssignmentSolver(
        self.spec_names, self.trans_names, self.equip_names, conn_st, conn_te
    )
    logs = self.solver_instance.solve()

    # 3. Виведення логу
    self.log_text.delete(1.0, tk.END)
    for l in logs:
        self.log_text.insert(tk.END, l + "\n")

    # 4. Ініціалізація візуалізації
    self.current_step_index = 0
    self.update_plot()
```

Рисунок 3.8 – Метод run_solver

```python
def update_plot(self):
    if not self.solver_instance or not self.solver_instance.history:
        return
    step_data = self.solver_instance.history[self.current_step_index]
    self.lbl_step.config(text=f"Етап: {step_data['iteration']} (Поріг > {step_data['threshold']})")
    self.ax.clear()
    G = nx.DiGraph()
    # Додаємо вузли
    for s in self.spec_names: G.add_node(s)
    for t in self.trans_names: G.add_node(t)
    for e in self.equip_names: G.add_node(e)
    edge_colors = []
    edge_widths = []
    edge_labels = {}
    # Додаємо ребра
    for (u, v), weight, is_active in step_data['all_edges']:
        G.add_edge(u, v)
        edge_labels[(u, v)] = str(weight)
        if is_active:
            edge_colors.append('red')
            edge_widths.append(2.5)
        else:
            edge_colors.append('lightgray')
            edge_widths.append(1.0)
    # Отримуємо красиві координати
    pos = self.calculate_positions()
    # Малюємо
    nx.draw_networkx_nodes(G, pos, ax=self.ax, node_color='lightblue', node_size=600)
    nx.draw_networkx_labels(G, pos, ax=self.ax)
    nx.draw_networkx_edges(G, pos, ax=self.ax, edge_color=edge_colors, width=edge_widths, arrows=True, arrowsize=15)
    # label_pos=0.35 зсуває підписи
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, ax=self.ax, font_size=8, label_pos=0.35)
    self.ax.set_title(f"Граф етапу {step_data['iteration']}\nЧервоним виділено активний потік (групи)")
    self.ax.axis('off')
    self.canvas.draw()
```

Рисунок 3.9 – Метод update_plot

Висновки за розділом 3

У третьому розділі представлено програмну реалізацію системи підтримки прийняття рішень для задачі про призначення на тридольному графі.

Обґрунтовано вибір мови програмування Python як ефективного інструменту для швидкої розробки програмного забезпечення. Використання спеціалізованих бібліотек NetworkX та Matplotlib дозволило реалізувати якісну візуалізацію графових структур, а бібліотека Tkinter забезпечила створення кросплатформного графічного інтерфейсу.

Описано структуру програмного продукту, який реалізує алгоритм пошуку максимального тривершинного поєднання з урахуванням нечітких ваг. Програмний код розділено на логічні модулі: обчислювальне ядро, що базується на методі Едмондса-Карпа та ітеративній фільтрації, та модуль інтерфейсу користувача.

Створений додаток дозволяє вводити вхідні дані довільної розмірності, автоматично формувати структуру транспортної мережі та виконувати пошук оптимального розподілу ресурсів. Реалізована система логування та покрокової візуалізації дає змогу досліднику детально аналізувати процес прийняття рішень на кожному етапі роботи алгоритму.

Отже, розроблений програмний засіб є повністю функціональним інструментом, готовим до проведення обчислювальних експериментів та перевірки ефективності запропонованої математичної моделі.

4 РЕЗУЛЬТАТИ ОБЧИСЛЮВАЛЬНОГО ЕКСПЕРИМЕНТУ ТА ЇХ АНАЛІЗ

4.1 Постановка задачі

Розглядається задача формування автономних рятувальних груп для дій у надзвичайних ситуаціях. Кожна група є трикомпонентною системою, що складається з одного спеціаліста, одного транспортного модуля та одного комплекту спеціального обладнання.

Нехай задано три скінченні множини об'єктів, що не перетинаються:

– $V_1 = \{S_1, S_2, ..., S_{n_1}\}$ – множина спеціалістів-рятувальників;

– $V_2 = \{T_1, T_2, ..., T_{n_2}\}$ – множина транспортних модулів;

– $V_3 = \{S_1, S_2, ..., S_{n_3}\}$ – множина комплектів обладнання.

Інформація про ефективність взаємодії між елементами цих множин має нечіткий характер. Тому введемо наступні характеристики.

Оцінка компетентності спеціаліста. Рівень володіння спеціалістом S_i навичками керування модулем T_j оцінюється за 100-бальною шкалою і задається трикутним нечітким числом (ТНЧ):

$$D_{ij}^{ST} = \left\langle a_{ij}^{ST}, b_{ij}^{ST}, c_{ij}^{ST} \right\rangle,$$

де a_{ij}^{ST} – песимістична оцінка компетентності;

b_{ij}^{ST} – найбільш очікувана оцінка компетентності;

c_{ij}^{ST} – оптимістична оцінка компетентності.

Функцію належності цього числа позначимо як $\mu_{D_{ij}^{ST}}(x)$.

Оцінка сумісності обладнання. Ефективність використання транспортного модуля T_j разом з комплектом обладнання E_k також оцінюється за 100-

бальною шкалою і описується трикутним нечітким числом:

$$D_{jk}^{TE} = \left\langle a_{jk}^{TE}, b_{jk}^{TE}, c_{jk}^{TE} \right\rangle.$$

Функцію належності позначимо як $\mu_{D_{jk}^{TE}}(x)$.

Введемо порогове значення вимог T. Вважатимемо, що взаємодія між об'єктами є надійною, якщо оцінка їх сумісності не нижча за T.

Ваги дуг графа w_{ij} та w_{jk} визначаються як ступінь впевненості у виконанні вимоги $x \geq T$. Тоді ступінь надійності w розраховується наступним чином:

$$w = \begin{cases} 1, & якщо\ b \geq T \\ \mu_D(T), & якщо\ b < T, c > T, \\ 0, & якщо\ \mathrm{c} \leq \mathrm{T} \end{cases} \tag{4.1}$$

де $\mu_D(T)$ розраховується за формулою правої гілки трикутного числа:

$$\mu_D(T) = \frac{c - T}{c - b}.$$

Необхідно знайти таке поєднання, яке б забезпечувало максимальний рівень надійності та максимальну кількість сформованих груп.

Згідно з постановкою задачі, маємо такий набір даних:

– спеціалісти V_1: S_1, S_2, S_3, S_4;

– транспортні модулі V_2: T_1, T_2, T_3;

– обладнання V_3: E_1, E_2, E_3, E_4.

Встановимо порогове значення вимог $T = 80$. Початкові нечіткі оцінки сумісності наведені у таблицях 4.1 та 4.2 (прочерк означає неможливість відпо-

відного призначення).

Таблиця 4.1 – Нечіткі оцінки зв’язків V_1 та V_2

	T_1	T_2	T_3
S_1	$\langle 70,78,90 \rangle$	$\langle 60,70,85 \rangle$	–
S_2	$\langle 55,74,90 \rangle$	$\langle 65,75,88 \rangle$	$\langle 50,60,70 \rangle$
S_3	$\langle 80,90,100 \rangle$	–	$\langle 72,79,92 \rangle$
S_4	$\langle 68,76,88 \rangle$	$\langle 74,81,94 \rangle$	$\langle 70,77,86 \rangle$

Таблиця 4.2 – Нечіткі оцінки зв’язків V_2 та V_3

	E_1	E_2	E_3	E_4
T_1	$\langle 70,79,95 \rangle$	$\langle 60,72,85 \rangle$	$\langle 65,75,82 \rangle$	–
T_2	$\langle 50,60,70 \rangle$	$\langle 75,85,95 \rangle$	$\langle 70,78,92 \rangle$	$\langle 60,70,85 \rangle$
T_3	–	$\langle 55,74,90 \rangle$	$\langle 68,76,90 \rangle$	$\langle 72,80,95 \rangle$

4.2 Хід роз’язання задачі

На рисунку 4.1 зображено тридольний граф, що відповідає умовам задачі. На ньому вершинам першої долі відповідають спеціалісти, вершинам другої долі – транспортні засоби, вершинам третьої долі – комплекти обладнання.

Розрахуємо ваги ребер на основі даних з таблиць 4.1 та 4.2. Для цього скористаємося формулою (4.1). Результати розрахунків наведено в таблицях 4.3 та 4.4.

На підготовчому етапі по графу G (рис. 4.1) будуємо граф G_1^*, що представлено на рисунку 4.2.

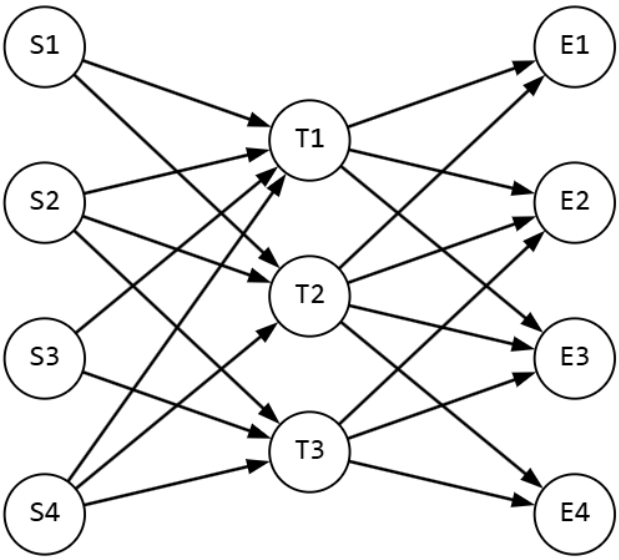

Рисунок 4.1 – Тридольний граф G

Таблиця 4.3 – Ваги дуг між вершинами V_1 та V_2

	T_1	T_2	T_3
S_1	$\frac{90-80}{90-78}=0.83$	$\frac{85-80}{85-70}=0.33$	–
S_2	$\frac{90-80}{90-74}=0.63$	$\frac{88-80}{88-75}=0.62$	0
S_3	1	–	$\frac{92-80}{92-79}=0.92$
S_4	$\frac{88-80}{88-76}=0.67$	1	$\frac{86-80}{86-77}=0.67$

Таблиця 4.4 – Ваги дуг між вершинами V_2 та V_3

	E_1	E_2	E_3	E_4
T_1	$\frac{95-80}{95-79}=0.94$	$\frac{85-80}{85-72}=0.38$	$\frac{82-80}{82-75}=0.29$	–
T_2	0	1	$\frac{92-80}{92-78}=0.86$	$\frac{85-80}{85-70}=0.33$
T_3	–	$\frac{90-80}{90-74}=0.63$	$\frac{90-80}{90-76}=0.71$	1

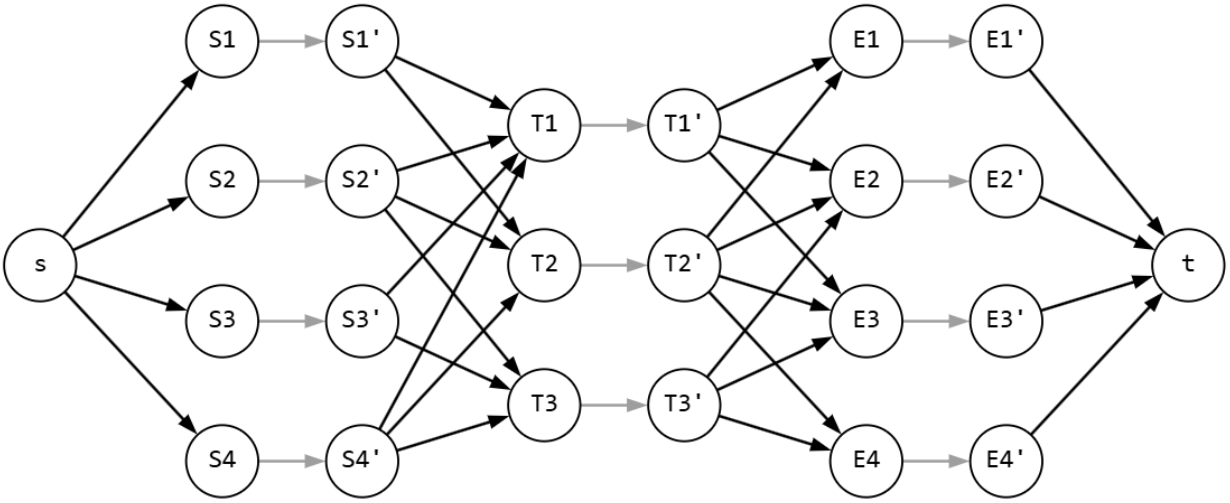

Рисунок 4.2 – Граф G_1^*, побудований на підготовчому етапі

На першому етапі для графа G_1^* будуємо максимальний потік (рис. 4.3), скориставшись методом Едмондса-Карпа. Величина максимального потоку v_1 дорівнює 3, можливими завантаженими шляхами можуть бути шляхи, що проходять через вершини:

$$s, S_1, S_1^{'}, T_1, T_1^{'}, E_1, E_1^{'}, t, s, S_2, S_2^{'}, T_2, T_2^{'}, E_2, E_2^{'}, t, s, S_3, S_3^{'}, T_3, T_3^{'}, E_3, E_3^{'}, t\ .$$

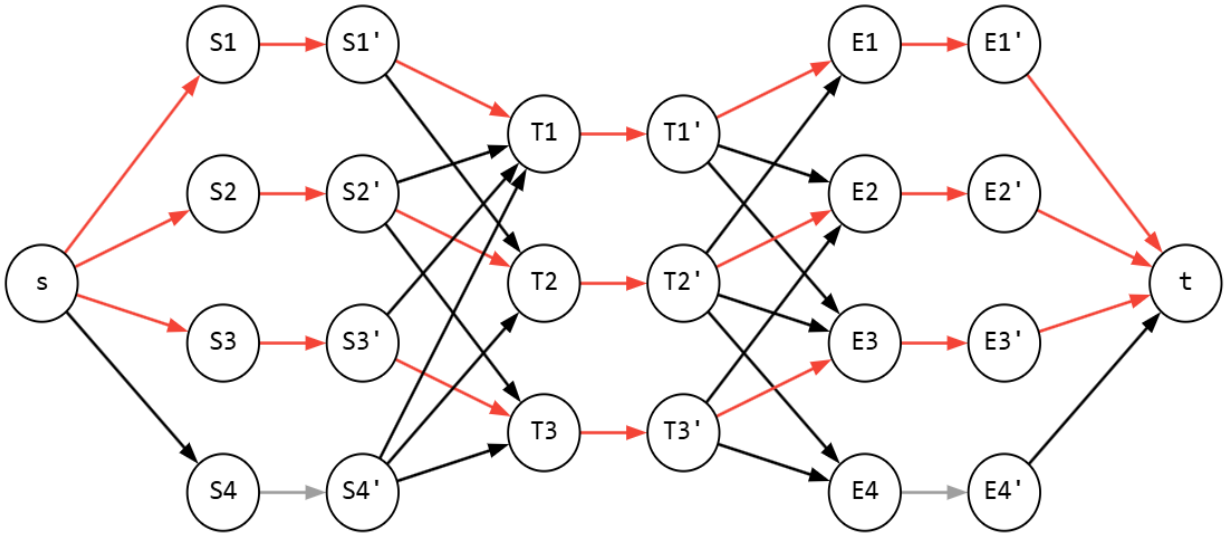

Рисунок 4.3 – Граф G_1^*, побудований на першому етапі,
завантажені дуги виділені червоними лініями

Їм відповідає тривершинне поєднання, що складається з трьох тривершинних ансамблів:

$$(S_1,T_1),\ (T_1,E_1),\ (S_2,T_2),\ (T_2,E_2),\ (S_3,T_3),\ (T_3,E_3).$$

Його вага τ_1 дорівнює 0,62. Прибираємо на графі G_1^* дуги, вага яких не перевищує 0,62. Отримуємо граф G_2^*, що зображений на рисунку 4.4.

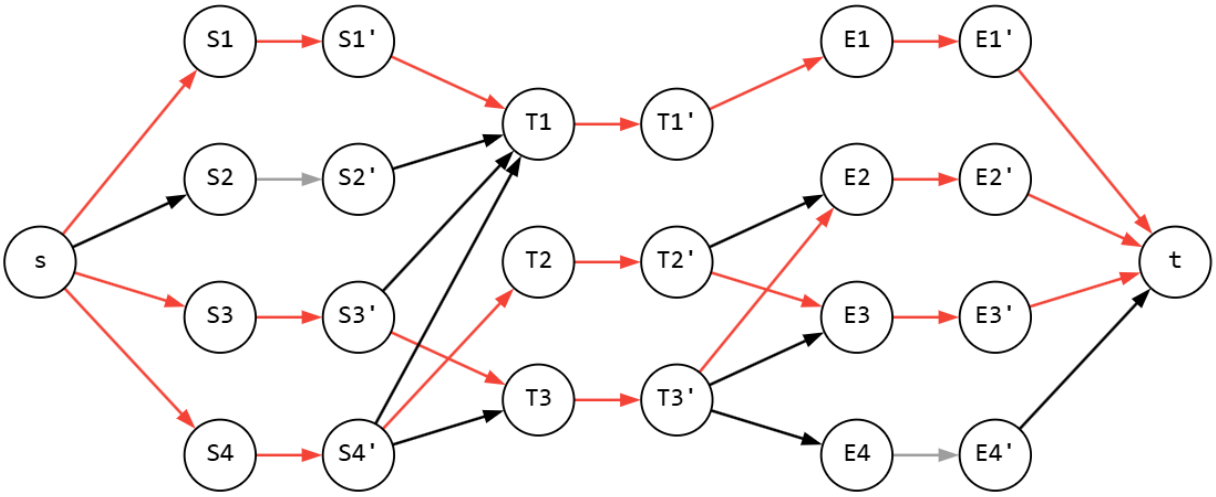

Рисунок 4.4 – Граф G_2^*, побудований на другому етапі

На другому етапі $v_2 = 3$, можливі завантажені шляхи на рисунку 4.4 виділені червоними лініями. Максимальне тривершинне поєднання складається з тривершинних ансамблів:

$$(S_1,T_1),\ (T_1,E_1),\ (S_3,T_3),\ (T_3,E_2),\ (S_4,T_2),\ (T_2,E_3).$$

Його вага τ_2 дорівнює 0,63. Прибираємо на графі G_2^* дуги, вага яких не перевищує 0,63. Отримуємо граф G_3^*, що зображений на рисунку 4.5.

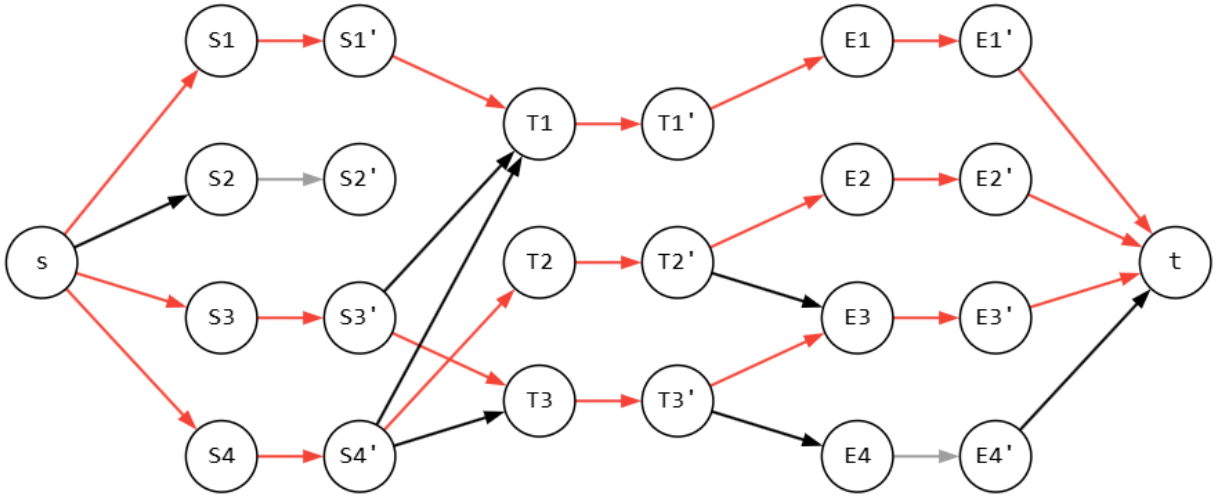

Рисунок 4.5 – Граф G_3^*, побудований на третьому етапі

На третьому етапі $v_3 = 3$, можливі завантажені шляхи на рисунку 4.5 виділені червоними лініями. Максимальне тривершинне поєднання складається з тривершинних ансамблів:

$$(S_1, T_1),\ (T_1, E_1),\ (S_3, T_3),\ (T_3, E_3),\ (S_4, T_2),\ (T_2, E_2).$$

Його вага τ_3 дорівнює 0,71. Прибираємо на графі G_3^* дуги, вага яких не перевищує 0,71. Отримуємо граф G_4^*, що зображений на рисунку 4.6.

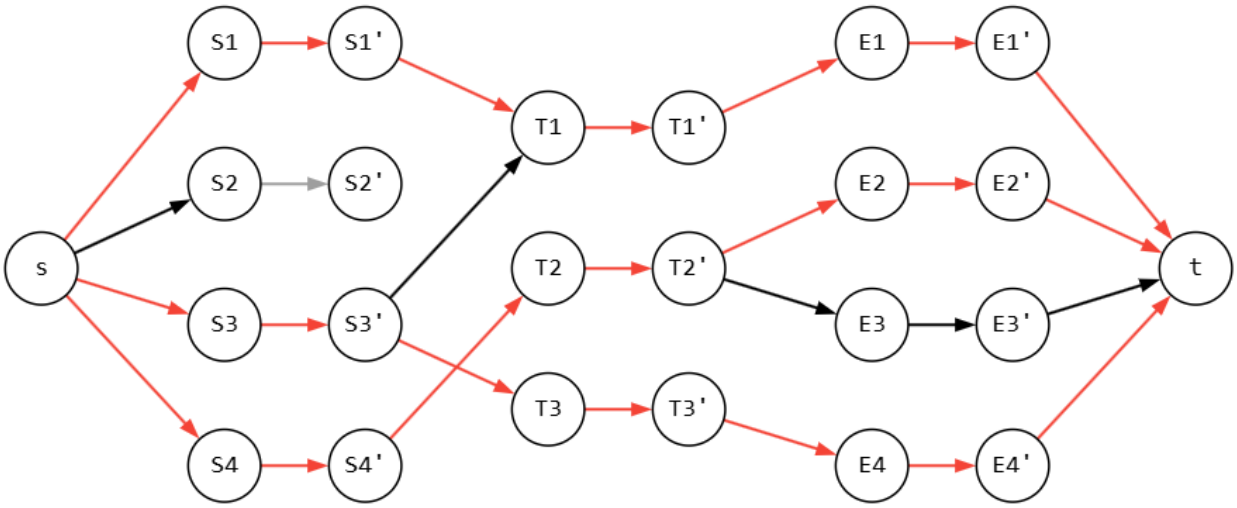

Рисунок 4.6 – Граф G_4^*, побудований на четвертому етапі

На четвертому етапі $v_4 = 3$, можливі завантажені шляхи на рисунку 4.6 виділені червоними лініями. Максимальне тривершинне поєднання складається з тривершинних ансамблів:

$$(S_1,T_1),\ (T_1,E_1),\ (S_3,T_3),\ (T_3,E_4),\ (S_4,T_2),\ (T_2,E_2).$$

Його вага τ_4 дорівнює 0,83. Прибираємо на графі G_4^* дуги, вага яких не перевищує 0,83. Отримуємо граф G_5^*, що зображений на рисунку 4.7.

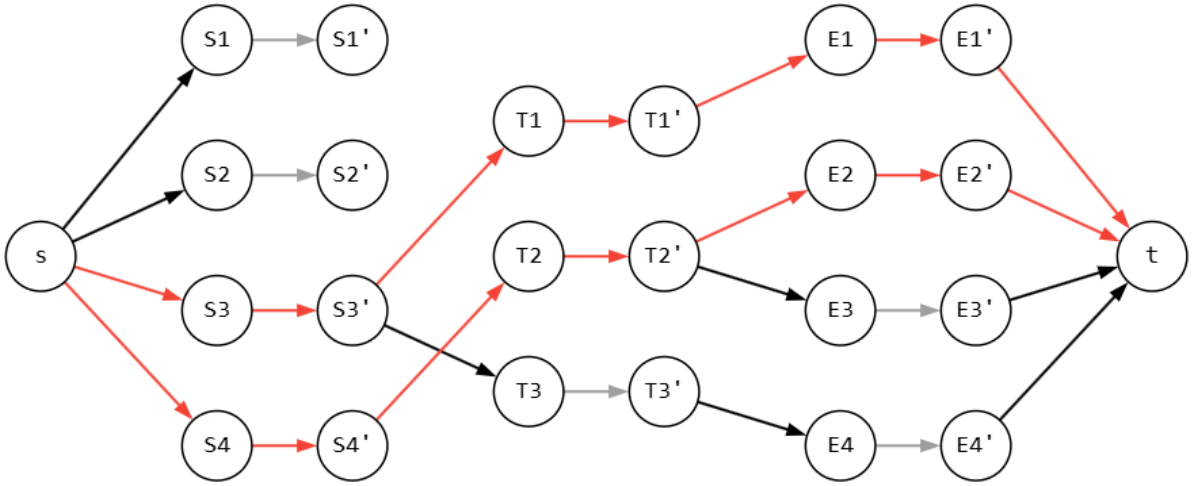

Рисунок 4.7 – Граф G_5^*, побудований на п’ятому етапі

На п’ятому етапі $v_5 = 2$, можливі завантажені шляхи на рисунку 4.7 виділені червоними лініями. Максимальне тривершинне поєднання складається з двох тривершинних ансамблів:

$$(S_3,T_1),\ (T_1,E_1),\ (S_4,T_2),\ (T_2,E_2).$$

Його вага τ_5 дорівнює 0,94. Бачимо, що надійність збільшилася, але потужність поєднання зменшилася.

Аналізуючи попередній етап, отримуємо відповідь: перший спеціаліст

займає перший транспортний засіб та бере перший комплект обладнання, третій спеціаліст займає третій транспортний засіб і бере четвертий комплект обладнання, четвертий спеціаліст займає другий транспорт і бере другий комплект обладнання.

Впевненість у тому, що спеціалісти будуть ефективно виконувати місії за допомогою обраних транспорту і обладнання, дорівнює 0.83.

4.3 Демонстрація роботи програми

Задамо параметри розмірів таблиць ваг та введемо вхідні дані задачі з таблиць 4.3 та 4.4 (рис. 4.8). При натисканні кнопки «Розв’язати задачу» ми отримуємо результати обчислень. Вони включають хід розв’язання задачі, що відображає виконання кожного етапу алгоритму, а також зображення графу на кожному етапі, який показує максимальний потік на поточному кроці і завантажені шляхи, що демонструють отримане тривершинне поєднання на цьому кроці.

На рисунках 4.9 – 4.13 представлені графи за кожним етапом розв’язку задачі. На рисунку 4.14 зображений хід розв’язання та результати розрахунків за кожним з етапів.

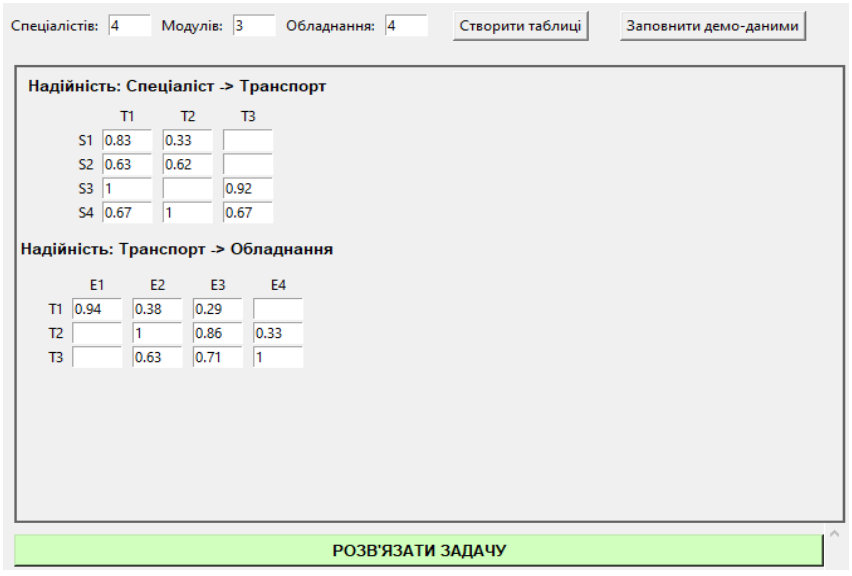

Рисунок 4.8 – Введення вхідних даних у програму

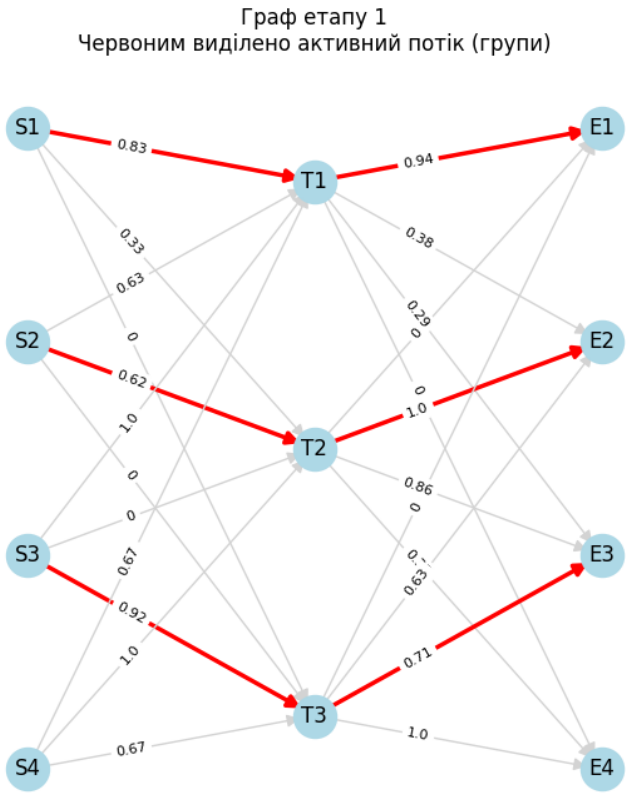

Рисунок 4.9 – Граф етапу 1

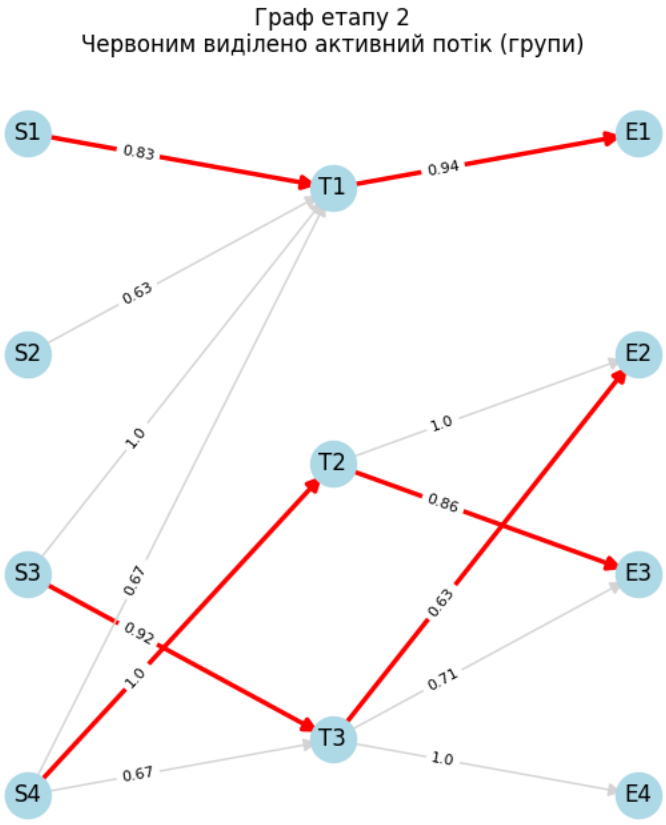

Рисунок 4.10 – Граф етапу 2

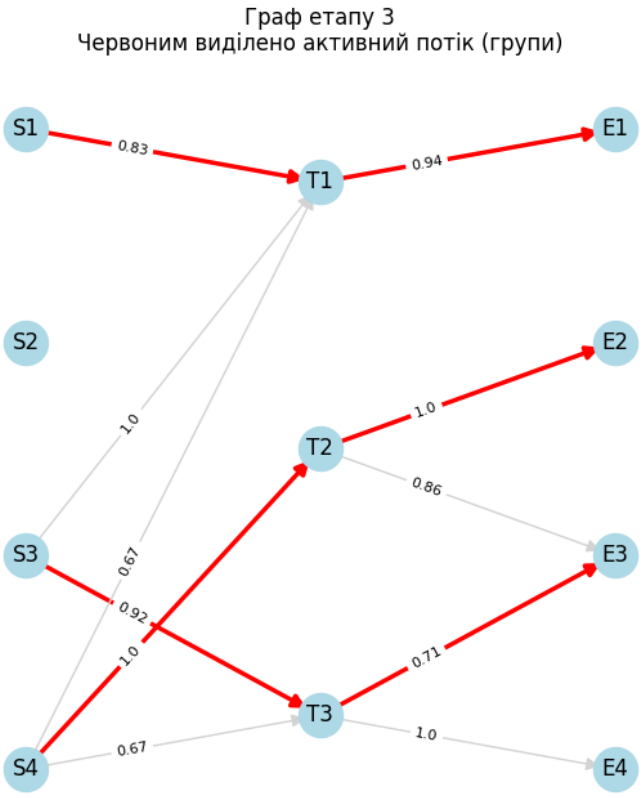

Рисунок 4.11 – Граф етапу 3

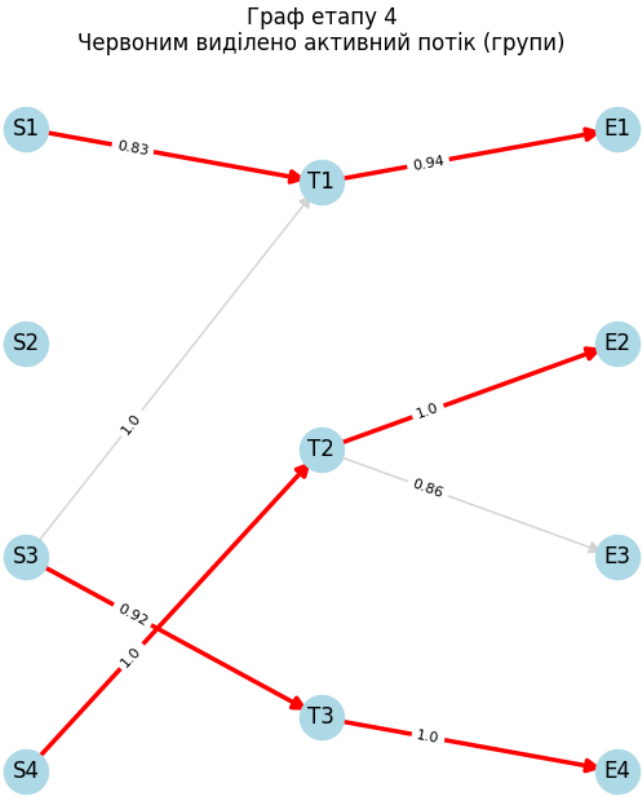

Рисунок 4.12 – Граф етапу 4

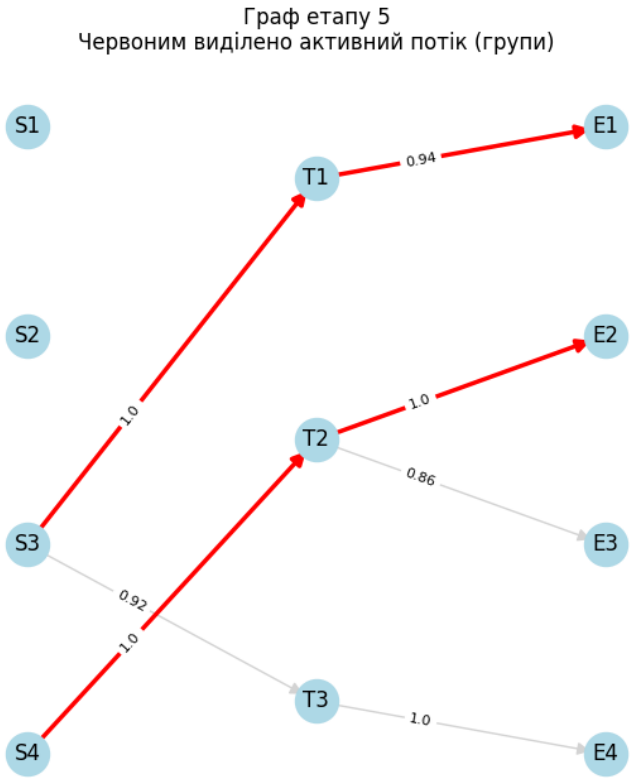

Рисунок 4.13 – Граф етапу 5

```
Хід розв'язання:
--- ЕТАП 1 (Поріг > -1.0) ---
Макс. потік (кількість груп): 3
Надійність системи: 0.62
Сформовані групи:
  S1 -> T1 -> E1 (Над: 0.83)
  S2 -> T2 -> E2 (Над: 0.62)
  S3 -> T3 -> E3 (Над: 0.71)

--- ЕТАП 2 (Поріг > 0.62) ---
Макс. потік (кількість груп): 3
Надійність системи: 0.63
Сформовані групи:
  S1 -> T1 -> E1 (Над: 0.83)
  S3 -> T3 -> E2 (Над: 0.63)
  S4 -> T2 -> E3 (Над: 0.86)

--- ЕТАП 3 (Поріг > 0.63) ---
Макс. потік (кількість груп): 3
Надійність системи: 0.71
Сформовані групи:
  S1 -> T1 -> E1 (Над: 0.83)
  S3 -> T3 -> E3 (Над: 0.71)
  S4 -> T2 -> E2 (Над: 1.0)

--- ЕТАП 4 (Поріг > 0.71) ---
Макс. потік (кількість груп): 3
Надійність системи: 0.83
Сформовані групи:
  S1 -> T1 -> E1 (Над: 0.83)
  S3 -> T3 -> E4 (Над: 0.92)
  S4 -> T2 -> E2 (Над: 1.0)

--- ЕТАП 5 (Поріг > 0.83) ---
Макс. потік (кількість груп): 2
Потік зменшився (2 < 3). ЗУПИНКА.
Оптимальне рішення було отримано на попередньому кроці.
```

Рисунок 4.14 – Хід розв’язання задачі

Як можна побачити, результати програмних обчислень збігаються із результатами обчислень, що були проведені в пункті 4.2.

Висновки за розділом 4

У четвертому розділі наведено результати обчислювального експерименту, метою якого була перевірка коректності роботи розробленого програмного забезпечення та практичне дослідження запропонованого алгоритму.

Сформульовано задачу комплектування рятувальних груп в умовах нечіткої інформації про сумісність компонентів (спеціалісти, транспортні модулі, обладнання). Проведено детальне розв’язання задачі з покроковим відображенням станів графа, що дозволило отримати еталонний результат.

Здійснено розв’язання ідентичної задачі за допомогою розробленої програми. Порівняльний аналіз результатів показав збіг вихідних даних з еталонними розрахунками, що підтверджує правильність програмної реалізації.

Продемонстровано можливості графічного інтерфейсу програми щодо візуалізації проміжних етапів розв’язання. Побудовані програмою графи та детальний звіт виконання алгоритму дозволяють досліднику наочно відстежувати процес відсіювання ненадійних зв’язків та динаміку формування оптимального розв’язку.

Таким чином, експериментально підтверджено, що розроблений програмний продукт є адекватним та ефективним інструментом для розв’язання задач про призначення на тридольних графах з нечіткими вершинами.

ВИСНОВКИ

У кваліфікаційній роботі розглянуто актуальне науково-прикладне завдання підвищення ефективності розподілу ресурсів у складних організаційних системах в умовах невизначеності. Шляхом системного аналізу та математичного моделювання досліджено метод комплектування трикомпонентних груп з використанням нечіткої логіки та теорії графів.

Проведено аналіз методів розв'язання задач про призначення. Обґрунтовано доцільність зведення задачі пошуку максимального тривершинного поєднання на нечіткому графі до задачі про максимальний потік у транспортній мережі. Запропоновано підхід до розрахунку ваг дуг на основі трикутних нечітких чисел.

Розроблено застосунок мовою Python з використанням графічного інтерфейсу, який дозволяє вводити дані довільної розмірності, візуалізувати структуру графа на кожному етапі оптимізації та отримувати детальний звіт про сформовані групи. Програмний продукт може бути адаптований до використання як система підтримки прийняття рішень (СППР) для керівників рятувальних служб, логістичних центрів та HR-відділів.

Впровадження дослідженого методу дозволяє підвищити надійність функціонування сформованих команд, що має прямий вплив на безпеку та ефективність ліквідації надзвичайних ситуацій. Оптимізація розподілу ресурсів сприяє зниженню витрат на експлуатацію техніки та підвищенню ефективності використання кадрового потенціалу.

Доцільним є продовження досліджень у напрямку узагальнення моделі на випадок k-дольних графів (де $k > 3$), що дозволить моделювати більш складні ланцюжки взаємодії. Також перспективним є врахування динаміки зміни нечітких параметрів у часі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Hoboken : Pearson, 2020. 1136 с.

2. Ertel W. Introduction to Artificial Intelligence. 3rd ed. Cham : Springer, 2021. 367 с.

3. Ощепков Є. С. Задача про призначення на тридольному графі з нечіткими вершинами. *29-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті»*: зб. матеріалів форуму (м. Харків, 16-19 квітня 2025 р.). Харків : ХНУРЕ, 2025. С. 303–304.

4. Hesamian G. Statistical Inference for Fuzzy Data. Cham : Springer, 2020. 309 с.

5. *Fuzzy Logic and Applications* : proceedings of WILF 2021, 13th Italian Workshop on Fuzzy Logic and Applications, Vietri sul Mare, Italy, December 20–22, 2021. Aachen : CEUR-WS.org, 2021. (CEUR Workshop Proceedings ; vol. 3074). URL: http://ceur-ws.org/Vol-3074/ (дата звернення: 21.11.2025).

6. *Solving Fuzzy Assignment Problems via Ranking Methods*: A Comprehensive Approach / S. Kumar, S. K. Suman, D. K. Sah [et al.]. International Journal of Scientific Research in Science and Technology. 2025. Т. 12, № 3. С. 877–889.

7. Матвієнко О. І., Закутній С. В. Нечітка логіка в задачах визначення економічних параметрів виконання проєктів. *Сучасний стан наукових досліджень та технологій в промисловості*. 2024. № 1 (27). С. 96–108.

8. Бардачов Ю. М., Соколова Н. А., Ходаков В. Є. Дискретна математика : підручник / за ред. В. Є. Ходакова. Київ : Вища шк., 2002. 287 с.

9. Bickle A. Fundamentals of Graph Theory. Boca Raton : CRC Press, 2020. 360 с.

10. Ус С. А., Коряшкіна Л. С. Моделі й методи прийняття рішень : навч. посіб. Дніпро : НГУ, 2014. 300 с.

11. Uthra G., Thangavelu K., Amutha B. An Approach of Solving Fuzzy Assignment Problem using Symmetric Triangular Fuzzy Number. *International Journal of Pure and Applied Mathematics*. 2017. Т. 113, № 7. С. 16–24.

12. Kaur P., Kumar A. A novel approach for solving fuzzy assignment problem with different types of fuzzy numbers. *International Journal of System Assurance Engineering and Management.* 2022. Т. 13, № 5. С. 2383–2398.

13. Khalaf W. S. A new method for solving fuzzy assignment problems. *Indonesian Journal of Electrical Engineering and Computer Science*. 2021. Т. 21, № 3. С. 1761–1768.

14. Kenneth C. R. Maximal fuzzy assignment problem involving symmetric hexagonal fuzzy number. *International Journal of Mathematics Trends and Technology. 2020.* Т. 66, № 5. С. 19–23.

15. A novel hybrid method using fuzzy decision making and multi-objective programming for sustainable-reliable supplier selection in two-echelon supply chain design / E. B. Tirkolaee, A. Mardani, Z. Dashtian [et al.]. *Journal of Cleaner Production*. 2020. Т. 250. Art. 119517.

16. Kochenderfer M. J., Wheeler T. A., Wray K. H. Algorithms for Decision Making. Cambridge, MA : MIT Press, 2022. 600 с.

17. Gentle J. E. Optimization. Cham : Springer, 2020. 598 с.

18. Korte B., Vygen J. Combinatorial Optimization: Theory and Algorithms. 7th ed. Berlin : Springer, 2024. 735 с.

19. Ford L. R., Fulkerson D. R. Maximal Flow Through a Network. *Canadian Journal of Mathematics*. 1956. № 8 (3). С. 399–404.

20. Introduction to Algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. 4th ed. Cambridge, MA : MIT Press, 2022. 1312 с.

21. Skiena S. S. The Algorithm Design Manual. 3rd ed. Cham : Springer, 2020. 810 с.

22. Python Software Foundation. Python 3.12.1 documentation. URL: https://docs.python.org/3.12/ (дата звернення: 28.11.2025).

23. Matthes E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. 3rd ed. San Francisco : No Starch Press, 2023. 552 с.

24. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming. 2nd ed. Sebastopol : O’Reilly Media, 2022. 1014 с.

25. Array programming with NumPy / C. R. Harris, K. J. Millman, S. J. van der Walt [et al.]. *Nature*. 2020. Т. 585. С. 357–362.

26. VanderPlas J. Python Data Science Handbook: Essential Tools for Working with Data. 2nd ed. Sebastopol : O’Reilly Media, 2022. 548 с.

27. Graphical User Interfaces with Tk. Python 3.12.1 documentation. URL: https://docs.python.org/3.12/library/tk.html (дата звернення: 29.11.2025).

28. NetworkX Developers. NetworkX 3.2.1 documentation. URL: https://networkx.org/documentation/stable/ (дата звернення: 28.11.2025).

29. Matplotlib Development Team. Matplotlib: Visualization with Python. URL: https://matplotlib.org/ (дата звернення: 29.11.2025).

30. Rougier N. P. Scientific Visualization: Python & Matplotlib. Bordeaux : Nicolas P. Rougier, 2021. 217 с.