Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

Магістерська робота
освітній ступінь – магістр

на тему: **«РОЗРОБКА ПАКЕТУ ВІЗУАЛІЗАЦІЇ ДИСКРЕТНИХ СТРУКТУР В ЗАДАЧАХ ОПТИМІЗАЦІЇ»**

Виконав: студент 2-го року навчання,
Спеціальності
121 Інженерія програмного забезпечення
Швачко Олег Русланович

Керівник Нагірна Алла Миколаївна
кандидат фізико-математичних наук, доцент
Рецензент Афонін Андрій Олександрович

Магістерська робота захищена
з оцінкою _____

Секретар ЕК _____
«__» _____ 2025 р.

Київ – 2025

Національний університет «Києво-Могилянська академія»

Факультет _____
Кафедра _____
Освітній ступінь _____
Спеціальність _____
Освітня/освітньо-наукова програма _____

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

«__» _____ 2025 року

З А В Д А Н Н Я
ДЛЯ КВАЛІФІКАЦІЙНОЇ / МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

1. Тема роботи

Розробка пакету візуалізації складних дискретних структур в задачах оптимізації

Керівник роботи

_____

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «» _____ 2024 року № ____

2. Строк подання студентом роботи

_____

3. План роботи

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ
1.1 Аналіз літератури і джерел з теми
1.2 Визначення основних понять і підходів
1.3 Постановка задачі оптимізації
РОЗДІЛ 2. ОПИС І РОЗРОБКА АЛГОРИТМІВ
2.1 Розробка архітектури програмного забезпечення
2.2 Опис обраних методів та алгоритмів
2.3 Підготовка до реалізації та тестування алгоритмів
2.4 Основи для реалізації дискретних задач оптимізації
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТИ
3.1 Опис інтерфейсу користувача.
3.2 Приклади використання програми
3.3 Аналіз результатів експериментів

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GUI (Graphical User Interface) – графічний інтерфейс користувача, що забезпечує взаємодію з програмою.

API (Application Programming Interface) – програмний інтерфейс для взаємодії між компонентами програмного забезпечення.

JSON (JavaScript Object Notation) – текстовий формат зберігання структурованих даних.

CSV (Comma-Separated Values) – формат для зберігання табличних даних.

XML (Extensible Markup Language) – формат обміну даними, що використовується для зберігання структурованої інформації.

OpenGL (Open Graphics Library) – програмний інтерфейс для рендерингу двовимірної та тривимірної графіки.

Dijkstra’s Algorithm – алгоритм пошуку найкоротшого шляху у зваженому графі з невід'ємними вагами.

A (A-star Algorithm)* – алгоритм пошуку найкоротшого шляху з використанням евристики.

Bellman-Ford Algorithm – алгоритм пошуку найкоротших шляхів у графі, що допускає від’ємні ваги ребер.

Prim’s Algorithm – алгоритм пошуку мінімального остовного дерева у зваженому графі.

BFS (Breadth-First Search) – алгоритм обходу графа в ширину.

DFS (Depth-First Search) – алгоритм обходу графа в глибину.

MST (Minimum Spanning Tree) – мінімальне остовне дерево, що містить усі вершини графа з мінімальною сумарною вагою ребер.

GLFW (Graphics Library Framework) – бібліотека для створення вікон OpenGL та взаємодії з користувачем.

ImGui (Immediate Mode GUI) – бібліотека для створення швидких графічних інтерфейсів.

ВСТУП

Сучасні тенденції розвитку інформаційних технологій активно стимулюють інновації у сфері аналізу, моделювання та оптимізації складних систем. Одним із ключових завдань залишається розробка ефективних програмних рішень для роботи зі складними дискретними структурами, такими як графи, дерева та ланцюги Маркова. Ці структури знаходять широке застосування в різних галузях, включаючи логістику, інформатику, економіку, системний аналіз та науку про дані. Візуалізація та інтерактивне управління цими структурами значно полегшують процес прийняття рішень, забезпечуючи швидкість і прозорість аналізу.

Актуальність роботи обумовлена стрімким збільшенням обсягів даних та ускладненням задач, які виникають у процесі їх аналізу та оптимізації. У більшості випадків методи, орієнтовані на текстовий або числовий аналіз, виявляються недостатньо ефективними для роботи зі складними дискретними структурами. Потреба у спеціалізованих програмних рішеннях, які дозволяють візуалізувати та інтерактивно керувати такими структурами, є очевидною. Зокрема, це важливо для задач пошуку найкоротших шляхів у графах, моделювання стохастичних процесів у ланцюгах Маркова, а також ребалансування дерев у задачах індексації [1; 2].

Об'єктом дослідження є складні дискретні структури, що використовуються для вирішення задач оптимізації в інформатиці та інших галузях. **Предметом дослідження** виступають методи візуалізації та інтерактивного управління дискретними структурами для підтримки процесів прийняття рішень.

Метою даної роботи є розробка універсального програмного пакету, який забезпечує ефективну візуалізацію та обчислювальні можливості для роботи зі складними дискретними структурами. Для досягнення цієї мети ставляться наступні завдання:

- аналіз сучасних підходів до моделювання та візуалізації дискретних структур [3];

- розробка архітектури програмного забезпечення, що забезпечує модульність і інтеграцію;

- імплементація основних алгоритмів візуалізації та оптимізації;

- тестування розробленого пакету на реальних даних.

Методологічна основа роботи включає об'єктно-орієнтоване програмування, аналіз алгоритмів та використання сучасних бібліотек для графічного інтерфейсу. Для реалізації використовувалися мови програмування Python і C++, а також бібліотеки OpenGL, Matplotlib та Tkinter [4; 5].

Наукова новизна роботи полягає в інтеграції сучасних методів оптимізації та візуалізації в єдиному програмному пакеті, що дозволяє як працювати з даними в реальному часі, так і інтерактивно демонструвати процеси аналізу. Практичне значення роботи полягає в можливості використання розробленого пакету для освітніх, наукових і прикладних цілей [6].

Для підготовки роботи було здійснено аналіз праць провідних вчених та практиків у галузі інформаційних технологій а також проведено аналіз існуючих програмних засобів та бібліотек коду python:

Захарова Л. В. «Інформаційні технології в освіті: монографія» [1].

Петренко О. В. «Сучасні тенденції розвитку програмного забезпечення для освіти» [2].

Hunter J. D. «Matplotlib: A 2D Graphics Environment» [3].

Документація Python Software Foundation (Python 3) [4].

ttkbootstrap: A modern theme for Tkinter [5].

SQLite Documentation [6].

Таким чином, розробка універсального програмного пакету для роботи зі складними дискретними структурами є важливим кроком для підтримки сучасних дослідників та інженерів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

1.1 Аналіз літератури і джерел з теми

Аналіз літератури та джерел, що стосуються візуалізації складних дискретних структур і задач оптимізації, виявив велику кількість наукових праць, які висвітлюють теоретичні та практичні аспекти теми. Серед цих робіт значну увагу привертають дослідження провідних вчених і розробки, здійснені в США, Німеччині, Японії, Великій Британії, Китаї та Канаді.

Огляд літературних джерел базується на монографіях, статтях, підручниках і технічній документації, які охоплюють широкий спектр тем — від основ дискретної математики до сучасних алгоритмів візуалізації та програмного забезпечення. Основна увага приділяється фундаментальним працям 1990-х років, які заклали теоретичний базис, а також сучасним дослідженням, що фокусуються на інтеграції штучного інтелекту та алгоритмів оптимізації. Визначальною рисою є географічне розмаїття авторів, включаючи роботи з США, Європи та Азії, що відображає глобальний характер теми. Проаналізовані ключові джерела інформації згруповані за типами і тематикою (рис 1.1).

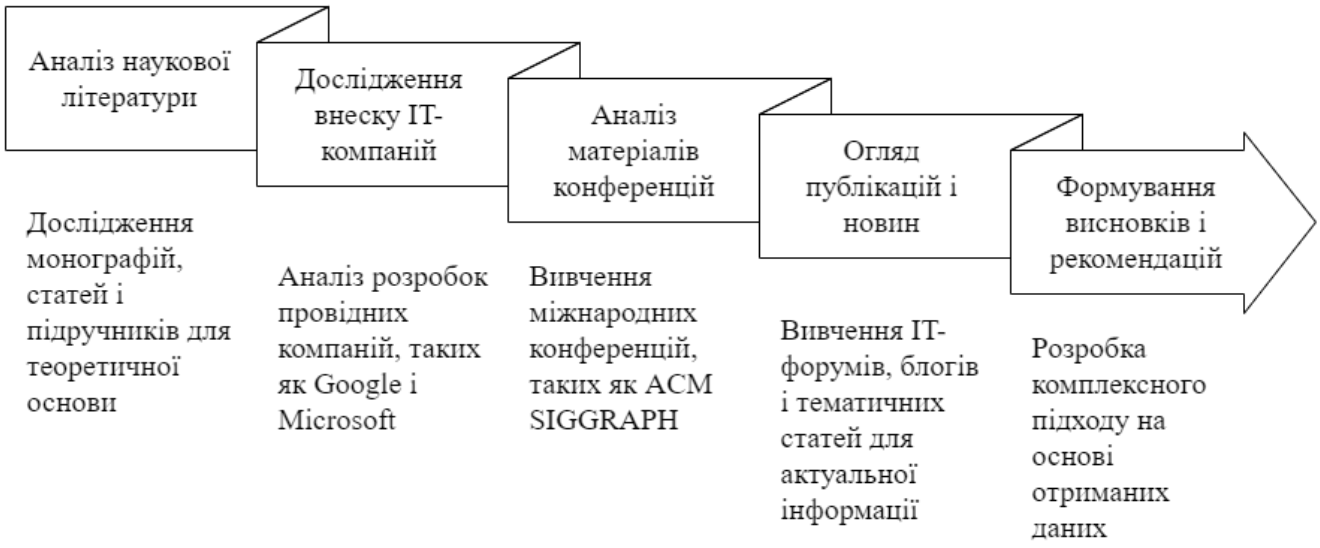


Рисунок 1.1 - Послідовність теоретичного дослідження

У монографії "Інформаційні технології в освіті" Л. В. Захарова [1] досліджується роль сучасних інформаційних технологій у розвитку навчального процесу та візуалізації складних процесів. Ця робота є важливою для розуміння принципів інтерактивного навчання, які можна адаптувати до задач оптимізації.

У статті О. В. Петренка [2] акцентовано увагу на важливості інтерактивності та естетики в розробці програмного забезпечення. Візуалізація складних структур, таких як графи або дерева, має бути не лише функціональною, але й зрозумілою для користувача, що особливо актуально для роботи з великими даними.

Робота Джона Хантера "Matplotlib: A 2D Graphics Environment" [3] описує підходи до створення графіків у Python, що є основою для побудови інструментів візуалізації дискретних структур. Використання бібліотек Matplotlib дозволяє адаптувати алгоритми для відображення графів або інших структур.

Книга "Graph Drawing: Algorithms for the Visualization of Graphs" (Di Battista, G. та ін., 1999) [7] є фундаментальною працею у сфері візуалізації графів. У ній детально розглядаються алгоритми розташування вузлів і ребер, що важливо для задач пошуку найкоротшого шляху або аналізу графових структур.

Документація Python [4] та SQLite [5] надає цінну інформацію щодо технічної реалізації алгоритмів та роботи з базами даних. Ці ресурси забезпечують стабільну основу для розробки інструментів оптимізації.

Використання бібліотеки ttkbootstrap [6] є прикладом сучасного підходу до створення графічних інтерфейсів. Вона дозволяє інтегрувати візуальні компоненти для кращого взаємозв’язку користувача з системою.

У книзі "Optimization in Operations Research" (Rardin R. L., 2017) [8] детально розглянуто математичні основи оптимізації, які є ключовими для алгоритмів, що працюють з дискретними структурами.

У підручнику "Discrete Mathematics and Its Applications" (Rosen K. H., 2019) [9] представлено теоретичні основи дискретної математики, включаючи графи, дерева та їх використання у різних задачах.

Робота "Efficient Graph-Based Algorithms" (Brandes U., Wagner D., 2003) [10] зосереджується на ефективності алгоритмів для роботи з графами. Автори з

Німеччини пропонують оптимізовані підходи для аналізу великих графів.

У статті "Visualization in Data Science: Techniques and Applications" (Wong P. C., Thomas J., 2016) [11] аналізуються сучасні підходи до візуалізації, включаючи інтерактивні інструменти для представлення складних процесів.

Дослідження "Deep Reinforcement Learning for Graph Optimization" (Wu Y., King I., 2021) [12] є прикладом інтеграції штучного інтелекту в задачі оптимізації, що може бути адаптоване для автоматизації візуалізації процесів.

Аналіз показав, що більшість досліджень у цій сфері зосереджена на створенні алгоритмів і платформ для роботи з графами, деревами, Марковськими ланцюгами тощо. Зокрема, в США та Німеччині виконуються передові дослідження у сфері алгоритмів візуалізації та оптимізації, тоді як Японія та Китай активно досліджують використання штучного інтелекту для вирішення подібних задач.

На основі огляду ключових джерел було виявлено, що сучасні підходи акцентуються на інтеграції алгоритмів оптимізації з інструментами візуалізації. У цьому контексті розробка універсального пакету для візуалізації складних дискретних структур не лише заповнює наявні прогалини, але й сприяє поширенню практичних рішень у цій галузі.

Також найбільш значну роль у розвитку технологій візуалізації дискретних структур відіграють провідні IT-компанії, які активно досліджують і розробляють нові рішення:

Google — компанія проводить дослідження у сфері алгоритмів оптимізації та візуалізації графів у проєктах на основі Google Maps, Knowledge Graph тощо. Їхні напрацювання включають використання методів машинного навчання для оптимізації графових структур.

Microsoft Research — дослідницький підрозділ Microsoft розвиває інструменти для візуалізації великих даних та алгоритмів оптимізації. Вагомий внесок зроблено у проєкти, пов’язані з обробкою графів і дерев, такі як Graph Engine, який дозволяє працювати з великими графами, забезпечуючи швидкий пошук і аналіз даних у розподілених середовищах.

IBM Research — працює над інноваціями у сфері моделювання та оптимізації

дискретних структур, інтегруючи штучний інтелект у задачі розв’язання найкоротших шляхів та аналізу великих графів. Зокрема, їхні дослідження включають розв’язання задачі найкоротшого шляху для великих графів і створення платформ для моделювання складних мереж.

Facebook (Meta) — значний фокус компанії спрямований на візуалізацію соціальних мереж як складних графів, аналіз їхніх структур та оптимізацію розподілених обчислень. Вони застосовують алгоритми для кластеризації користувачів, прогнозування зв’язків і оптимізації розподілених обчислень для обробки великих графів.

NVIDIA — досліджує паралельні алгоритми для візуалізації складних структур, зокрема з використанням технологій GPU. Їхня бібліотека cuGraph забезпечує високопродуктивне виконання операцій над графами в реальному часі, включаючи обчислення центральності, пошук шляхів та кластеризацію.

Oracle — активно використовує технології роботи з графами для оптимізації баз даних та їхньої візуалізації, зокрема в Oracle Spatial and Graph.

Важливу роль у популяризації та обговоренні нових досліджень відіграють міжнародні конференції, які об'єднують провідних науковців і розробників:

ACM SIGGRAPH (Special Interest Group on Computer Graphics and Interactive Techniques) — конференція, яка фокусується на новітніх розробках у сфері графіки та інтерактивних технологій.

IEEE Visualization (VIS) — конференція зосереджена на методах візуалізації даних, зокрема складних структур, таких як графи та дерева.

International Symposium on Graph Drawing and Network Visualization (GD) — спеціалізована подія, присвячена виключно методам візуалізації графів.

NeurIPS (Neural Information Processing Systems) — хоча головною темою є машинне навчання, багато досліджень торкаються візуалізації графів і дерев у контексті алгоритмів глибокого навчання.

International Conference on Machine Learning (ICML) — розглядає сучасні підходи до роботи з графовими структурами та їхньої оптимізації.

Ці компанії та конференції забезпечують не лише науковий, але й практичний

фундамент для розробки нових технологій у сфері оптимізації та візуалізації дискретних структур. Їхні досягнення є невід'ємною частиною сучасних досліджень і розробок у даній галузі.

1.2 Визначення основних понять і підходів

Розробка пакету візуалізації дискретних структур у задачах оптимізації вимагає чіткого розуміння ключових понять та підходів, які забезпечують фундамент для створення ефективних інструментів. Дискретні структури, такі як графи, дерева та ланцюги Маркова, відіграють важливу роль у задачах оптимізації. Візуалізація цих структур сприяє кращому розумінню процесів та прийняттю рішень. Основні підходи включають як алгоритмічні рішення для роботи зі структурами, так і технологічні методи реалізації візуалізації. В таблиці 1.1 визначаються базові концепції, що лежать в основі роботи, з подальшим аналізом їх застосування на практиці.

Таблиця 1.1 - Основні поняття та підходи досліджуваної теми

Категорія	**Елемент**	**Опис**
Дискретні структури	Графи	Вершини та ребра, що моделюють мережі, маршрути, зв'язки.
	Дерева	Ієрархічні структури без циклів.
	Ланцюги Маркова	Стохастичні моделі переходів між станами.
Оптимізація	Жадібні алгоритми	Алгоритми, що шукають локально оптимальні рішення (наприклад, Дейкстра).
	Динамічне програмування	Розбиття задачі на підзадачі для оптимізації обчислень.
	Генетичні алгоритми	Ітеративний пошук рішень, що імітує еволюційні процеси.
Візуалізація	Статична	Графіки та схеми, які показують підсумкові дані.
	Динамічна	Інтерактивне відображення процесів і алгоритмів.
Технології	OpenGL	Бібліотека для високопродуктивної графіки.
	Python (Matplotlib, NetworkX)	Інструменти для розробки та візуалізації алгоритмів.
	WebGL	Технологія для інтерактивної візуалізації в браузері.

Розглянемо приклади для кожного наведеного елементу:

Графи: мережа доріг у місті, де вершини — це перехрестя, а ребра — дороги. Алгоритм Дейкстри може бути використаний для пошуку найкоротшого маршруту між двома перехрестями. На рисунку 1.2 червоні ребра позначають найкоротший шлях, знайдений алгоритмом, а ваги ребер відображають відповідні значення. Це є приклад динамічної візуалізації процесу в задачах оптимізації.

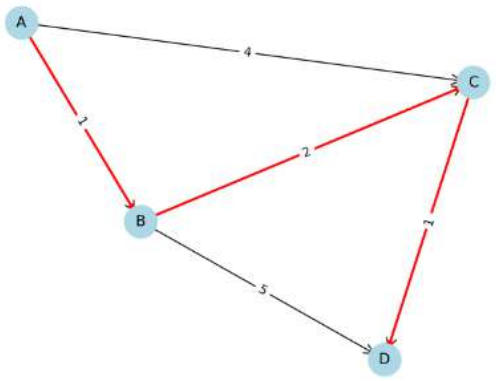


Рисунок 1.2 - Алгоритму Дейкстри для пошуку найкоротшого шляху

Дерева: ієрархічна структура категорій товарів в інтернет-магазині. Наприклад, "Електроніка" → "Смартфони" → "Аксесуари" (рис. 1.3). Вузли показують основні категорії та підкатегорії: від "Електроніка" до більш деталізованих груп, таких як "Смартфони" → "Аксесуари". Ця структура може бути корисною для моделювання ієрархій або пошуку відповідних товарів у системах управління даними.

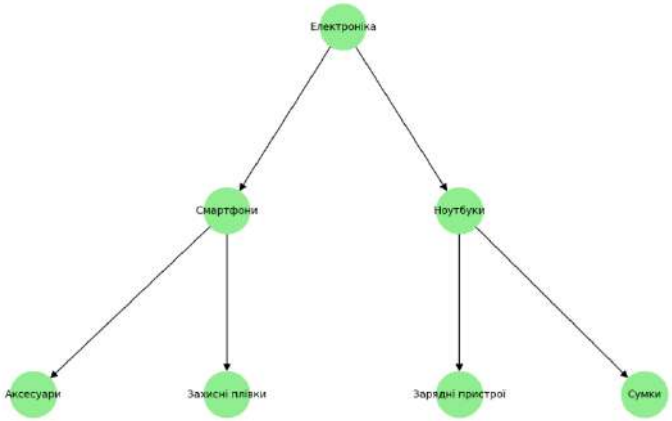


Рисунок 1.3 - Ієрархічна структура категорій товарів

Ланцюги Маркова: модель переходів між станами клієнтів в онлайн-магазині: "Відвідувач" → "Додав у кошик" → "Здійснив покупку". На рисунку 1.4 показано, як з часом змінюється ймовірність перебування клієнтів у кожному стані. Початково всі клієнти є "Відвідувачами", але з часом частина з них переходить у стан "Додав у кошик", потім "Здійснив покупку", а решта — залишає сайт. Це наочний приклад, який може бути використаний для прогнозування поведінки клієнтів та оптимізації роботи онлайн-магазину.

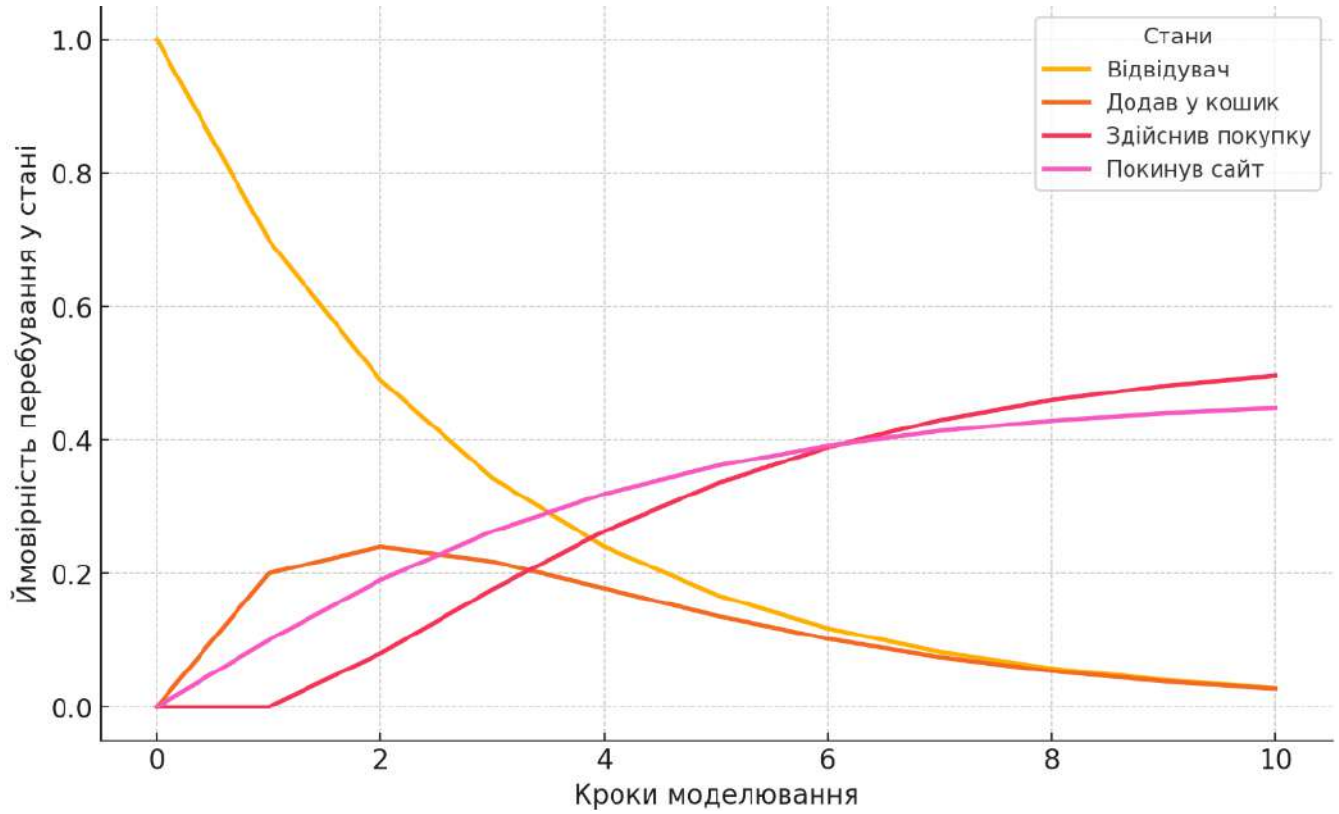


Рисунок 1.4 - Еволюція станів клієнтів у моделі ланцюгів Маркова

Жадібні алгоритми: Оптимізація маршруту доставки для кур'єрів, де кожен наступний крок мінімізує загальну відстань. Рисунок 1.5 демонструє вузли, які представляють склад і клієнтів, а ребра — дороги між ними з вагами, що відповідають відстаням.

Починаючи з вузла "Склад" на кожному кроці вибираємо найближчого клієнта (вузол), якого ще не відвідали, повторюємо, доки всі клієнти не будуть відвідані.

В результаті червоні ребра показують маршрут, який мінімізує загальну

відстань. У підсумку отримано ефективний шлях, який можна використовувати для оптимізації логістики доставки. Даний тип алгоритму активно використовується в розробці та обговорюється на IT формумах, зокрема на українському форумі https://devzone.org.ua [13].

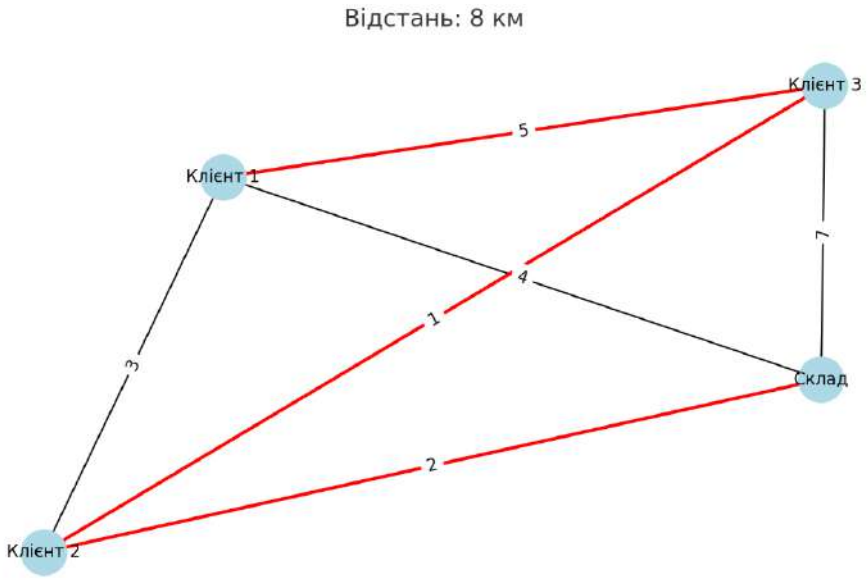


Рисунок 1.5 - Жадібний алгоритм з оптимізацією маршруту доставки

Динамічне програмування: розрахунок оптимального розкладу занять у школі з урахуванням кількох обмежень (вільні кабінети, час викладачів). Цей підхід до розв’язання задач передбачає розбиття складної задачі на простіші підзадачі з повторним використанням їхніх результатів [14, 15]. Динамічне програмування дозволяє розраховувати оптимальний розклад занять у школі з урахуванням обмежень, таких як доступність кабінетів і графік викладачів. У задачі вузли відповідають урокам, стан визначає вже розміщені уроки з відповідними параметрами, а перехід передбачає додавання нового уроку. Завдяки рекурсивному підходу з мемоізацією зберігаються проміжні рішення для уникнення повторних обчислень. Вартість рішення враховує порушення обмежень, наприклад, конфлікти у використанні кабінетів чи викладачів. Такий підхід забезпечує ефективний і масштабований спосіб оптимізації розкладу. В таблиці 1.2 показано модель розкладу занять, яка враховує доступність вчителів, кабінетів та часових слотів.

Таблиця 1.2 - модель розкладу занять з врахуванням доступності об’єктів

Урок	**Вчитель**	**Кабінет**	**Часовий слот**
Math	T1	R1	9:00-10:00
Physics	T2	R2	9:00-10:00
Chemistry	T1	R1	10:00-11:00
Biology	T2	R2	10:00-11:00

Генетичні алгоритми: оптимізація розташування складів для мінімізації логістичних витрат. Рішення кодуються як набори складів, оцінюється сумарна відстань до клієнтів, після чого найкращі варіанти комбінуються та змінюються. У підсумку отримується конфігурація складів, що забезпечує мінімальні логістичні витрати.

Статична візуалізація: побудова діаграми розподілу товарів за категоріями. На рисунку 1.6 показано як на основі даних про кількість товарів у різних категоріях (наприклад, "Продукти харчування", "Напої", "Електроніка") побудована стовпчаста діаграма, яка дозволяє легко порівняти популярність або розподіл товарів між категоріями.

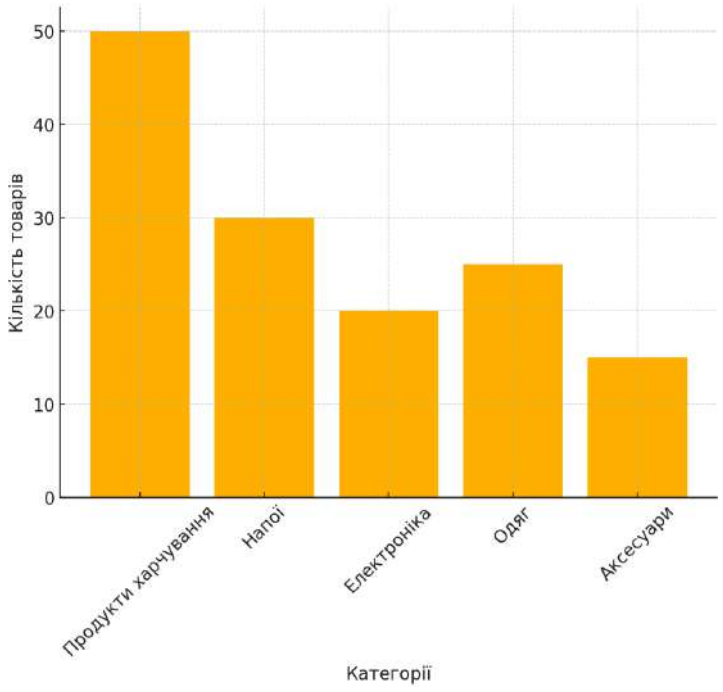


Рисунок 1.6 - Розподіл товарів за категоріями

Динамічна візуалізація: інтерактивне відображення процесу пошуку

найкоротшого шляху в графі. Інтерактивна візуалізація демонструє процес роботи алгоритму Дейкстри на графі, показуючи, як поступово обираються вузли та оновлюються найкоротші шляхи до кожного з них. У реальному часі користувач може спостерігати, як алгоритм переходить від одного вузла до іншого, підсвічуючи обрані ребра та відображаючи проміжні результати. На рисунку 1.7 зображено динамічна візуалізація, яка показує процес пошуку найкоротшого шляху в графі за допомогою алгоритму Дейкстри. На кожному кроці підсвічуються ребра, які додаються до найкоротшого шляху, аж до його завершення.

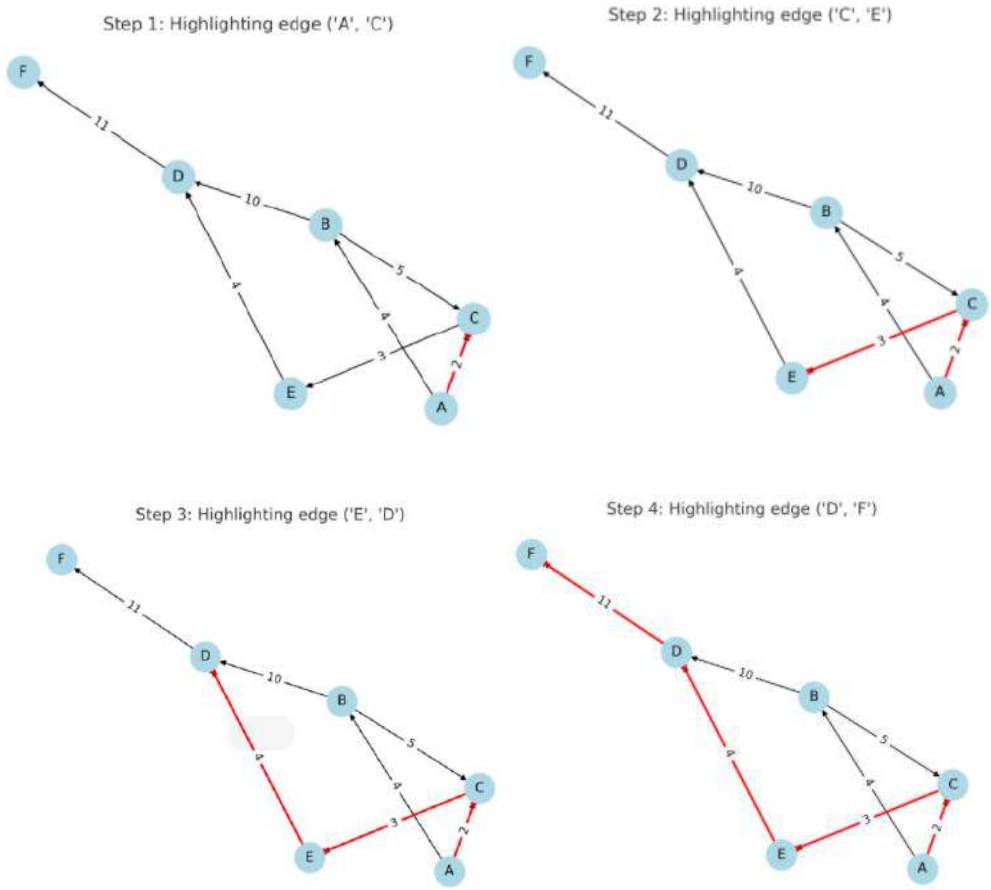


Рисунок 1.7 - Демонстраця етапів пошуку найкоротшого шляху

OpenGL використовується для створення візуалізації 3D-моделей, наприклад, дерева рішень для задачі класифікації, що дозволяє чітко уявити ієрархію та взаємозв’язки між класами [16]. Python, зокрема його бібліотеки Matplotlib і NetworkX, є ефективними інструментами для побудови графіків і моделювання графів, що ілюструють зростання складності алгоритмів у залежності від кількості

вузлів [3; 17]. WebGL забезпечує інтерактивне моделювання мережевих графів у веб-додатках, що використовується, наприклад, для візуалізації топології Інтернет-з'єднань, надаючи користувачам динамічний спосіб аналізу структури [18].

1.3 Постановка задачі оптимізації

Метою оптимізації є пошук найкращого рішення серед множини можливих варіантів з урахуванням заданих критеріїв та обмежень. У контексті даного дослідження це означає ефективну роботу з дискретними структурами, такими як графи, дерева та ланцюги Маркова, для вирішення задач, що потребують мінімізації витрат, часу або ресурсів, або ж максимізації певних показників [8; 9; 14].

Оптимізація дозволяє не лише знаходити точні або наближені рішення складних задач, але й покращувати ефективність процесів, що є важливим для практичного застосування в різних галузях, таких як логістика, інформаційні системи, аналіз даних і розподілені обчислення. У цьому дослідженні акцент робиться на інтеграції оптимізаційних алгоритмів із візуалізацією для спрощення аналізу та прийняття рішень [11; 12].

Оптимізація дискретних структур є ключовою задачею в сучасних науках про дані, інформатиці та операційному менеджменті. У світі зростаючого обсягу інформації та складності систем ефективне вирішення задач оптимізації стає все більш актуальним. Дискретні структури, такі як графи, дерева та ланцюги Маркова, широко застосовуються для моделювання складних систем у логістиці, мережевому аналізі, управлінні даними та штучному інтелекті.

Зростання вимог до швидкості обробки інформації та прийняття рішень робить необхідним використання ефективних алгоритмів і підходів до оптимізації. Зокрема, інтеграція візуалізації з алгоритмами оптимізації дозволяє не лише автоматизувати процес пошуку рішень, але й робити його більш прозорим та зрозумілим для користувачів. Це відкриває нові можливості для взаємодії людини з даними, особливо у галузях, де потрібна швидка адаптація до змін або аналіз великих обсягів інформації.

Дослідження у цій сфері має важливе значення для розвитку практичних застосувань, таких як оптимізація транспортних маршрутів, ефективне управління ресурсами, проектування інформаційних систем та інтелектуальних алгоритмів для моделювання складних процесів. Таким чином, задача оптимізації дискретних структур має як теоретичну, так і практичну актуальність, сприяючи розвитку інструментів для вирішення широкого спектра реальних проблем.

Об’єктом оптимізації є складні дискретні структури, які використовуються для моделювання реальних процесів і систем, таких як транспортні мережі, ієрархії даних, а також ймовірнісні системи. Ці структури, зокрема графи, дерева та ланцюги Маркова, є основою багатьох задач у сфері оптимізації. Предметом оптимізації є алгоритми, які дозволяють вирішувати ці задачі, забезпечуючи мінімальні витрати ресурсів, часу або інших критеріїв. Оптимізація таких структур має широкий спектр застосувань, включаючи логістику, аналіз даних, управління ресурсами та моделювання динамічних систем.

Оптимізація дискретних структур вимагає чіткого визначення вхідних даних і параметрів, які впливають на результати. Вхідні дані можуть варіюватися залежно від типу структури, але завжди охоплюють базові характеристики структури, обмеження та критерії (табл. 1.3).

Таблиця 1.3 - Вхідні дані та параметри

Структура	**Вхідні дані**	**Параметри**
Графи	Вузли та ребра (із вагами), стартовий та кінцевий вузли	Максимальна кількість вузлів, обмеження на ваги ребер
Дерева	Вузли, зв’язки між ними	Критерії балансу (глибина, симетрія), оптимізація за ефективністю пошуку
Ланцюги Маркова	Стани системи, ймовірності переходів між станами	Початковий стан, кількість ітерацій
Загальні	Інтерактивно введені дані користувачем, обмеження ресурсів	Часові обмеження, можливість моделювання в реальному часі

Оптимізація кожного типу дискретних структур має свої специфічні цілі, але загалом очікується досягнення рішень, які мінімізують витрати або максимізують ефективність системи (табл. 1.4).

Таблиця 1.4 - Таблиця очікуваних результатів

Структура	**Очікуваний результат**
Графи	Найкоротший шлях, мінімальна остова структура або оптимальний розподіл потоків
Дерева	Збалансована структура дерева з мінімальною глибиною або оптимізацією пошукових операцій
Ланцюги Маркова	Ймовірності перебування в кожному стані після певної кількості ітерацій
Загальні	Інтерактивне візуальне представлення оптимізованого рішення з можливістю аналізу процесу

Формулювання задачі оптимізації дозволяє чітко визначити об’єкт і предмет дослідження, що забезпечує основу для реалізації ефективних алгоритмів. Врахування специфічних вхідних даних і параметрів для кожної структури дає змогу адаптувати підходи до різних умов. Очікувані результати оптимізації підкреслюють важливість інтеграції алгоритмів і візуалізації для вирішення практичних завдань у різних галузях.

Задачі оптимізації класифікуються залежно від типу об’єктів, що оптимізуються, та критеріїв, які потрібно досягти. Розглянути різні класи задач, що охоплюють дискретну, комбінаторну, ймовірнісну та мультиоб’єктну оптимізацію. Кожен із цих класів має свою специфіку, методи розв’язання та області застосування. Їхнє розуміння та правильна класифікація дозволяють ефективно адаптувати підходи до вирішення широкого спектра практичних задач, пов’язаних із дискретними структурами (рис. 1.8).

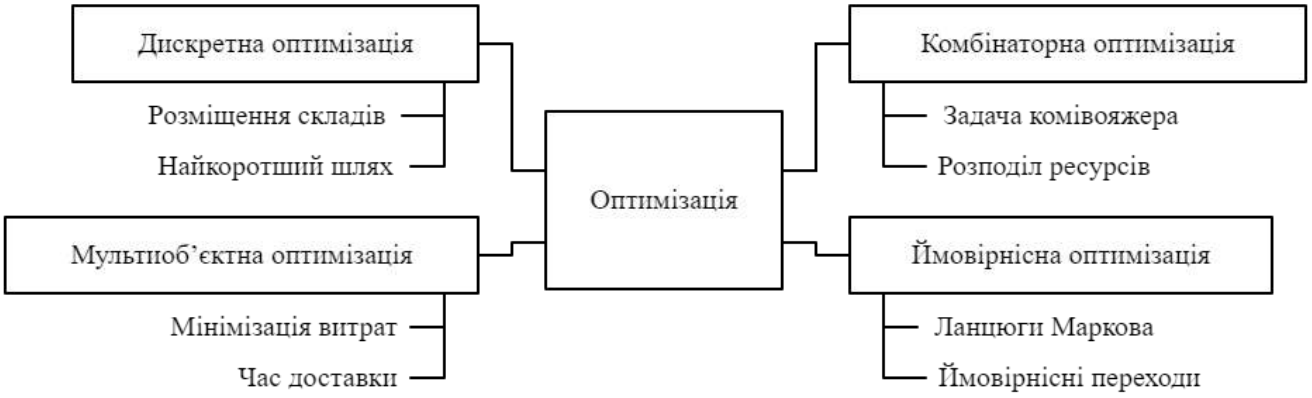


Рисунок 1.8 - Огляд типів задач

Особливості та специфіка кожного типу задач:

1. При дискретній оптимізації рішення обирається із заздалегідь визначеного набору варіантів. Алгоритми часто вимагають перебору можливих комбінацій, що робить проблему складною для великих множин.

2. В комбінаторній оптимізації є фокус на пошуку найкращої комбінації елементів, часто використовуючи евристики. Проблеми цього класу зазвичай належать до класу NP-важких задач, що потребують спеціалізованих алгоритмів для ефективного розв’язання (жадібні алгоритми, генетичні алгоритми тощо).

3. У ймовірнісній оптимізації все залежить від статистичних властивостей системи. Вона вимагає врахування невизначеності даних та застосування моделей прогнозування.

4. Мультиоб’єктна оптимізація пропонує рішення, що має балансувати між кількома суперечливими критеріями. В ній використовуються методи Парето-оптимізації або багатокритеріального аналізу для пошуку компромісних рішень.

Обмеження в задачах оптимізації визначають набір умов, які повинні виконуватись для допустимих рішень, наприклад:

- ресурсні обмеження на кількість доступних ресурсів, таких як час, пам'ять або потужність обчислювальних систем. Наприклад, у задачі оптимізації маршруту доставка повинна завершитись у заданий часовий проміжок;

- умови структур, тобто особливості структури даних, зокрема зв'язність графів або збалансованість дерев. Наприклад, у графі для задачі найкоротшого шляху всі ребра повинні мати додатні ваги;

- обмеження на рішення - що описують вимоги щодо допустимих рішень, наприклад, вибір підмножини елементів, яка задовольняє задані критерії. У ланцюгах Маркова може бути обмеження на кількість переходів між станами.

В совю чергу критерії оптимальності визначають, наскільки ефективним є запропоноване рішення, і служать базою для вибору найкращого варіанту::

- мінімізація витрат - мінімальна відстань у задачі пошуку найкоротшого шляху;

- максимізація ефективності - максимізація пропускної здатності мережі у

графах;

- баланс між показниками - для мультиоб’єктних задач важливо знайти компроміс між кількома критеріями, наприклад, мінімізація часу та витрат;

- стабільність рішень - в задачах, що залежать від випадковості (наприклад, у ланцюгах Маркова), важливо забезпечити стабільність результатів навіть за умов зміни початкових даних.

Обмеження визначають рамки задачі, тоді як критерії оптимальності дозволяють обрати найкраще рішення з множини можливих. Їх правильне поєднання є критично важливим для успішного вирішення задач оптимізації, особливо в умовах роботи з дискретними структурами.

Оптимізація дискретних структур вимагає застосування різноманітних методів, які дозволяють ефективно вирішувати задачі з різними характеристиками складності. Методи можна класифікувати на алгоритмічні, евристичні та гібридні, кожен із яких має свої переваги і недоліки залежно від специфіки задачі (рис. 1.9) .

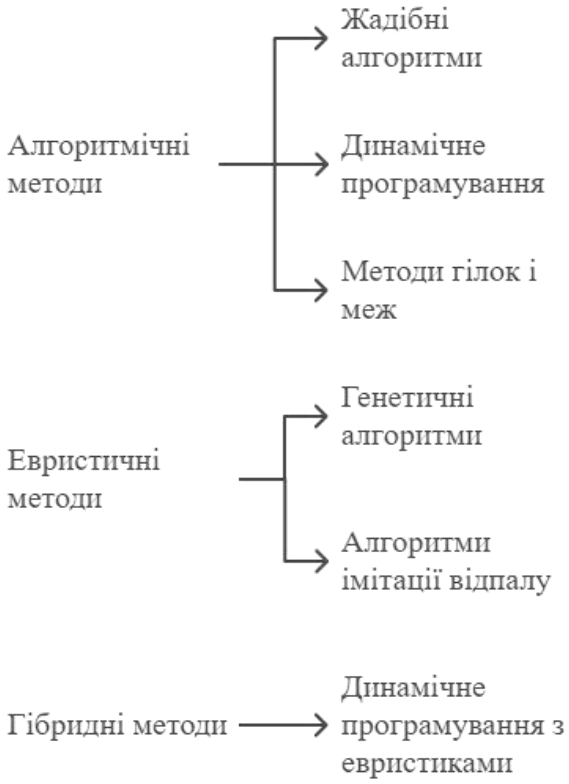


Рисунок 1.9 - Методи оптимізації дискретних структур

Залежно від характеристик задачі, доцільність використання того чи іншого методу обумовлюється наступними факторами:

1. Складність задачі. NP-важкі задачі потребують евристичних або гібридних підходів для скорочення часу розрахунків і отримання наближених рішень.

2. Тип структури. Для задач на графах і деревах алгоритмічні методи (жадібні алгоритми, динамічне програмування) є найбільш ефективними завдяки чіткій структурі даних.

3. Невизначеність. У задачах з невизначеними параметрами (наприклад, ланцюги Маркова) застосовуються ймовірнісні та евристичні підходи.

4. Масштабність. Гібридні методи забезпечують гнучкість і масштабованість для задач із великими обсягами даних, таких як розподіл ресурсів або транспортні задачі.

Різноманіття методів оптимізації дозволяє адаптувати підходи до специфіки задачі, забезпечуючи ефективність і точність. Вибір методу визначається характером дискретної структури, обмеженнями та вимогами до результату. Такий підхід дозволяє досягти оптимальних результатів для широкого спектра задач.

Оптимізація складних задач передбачає їх розбиття на менші підзадачі, які є простішими для аналізу та вирішення. Такий підхід дозволяє структурувати процес розв’язання, зменшити обчислювальну складність і забезпечити взаємозв’язок між етапами роботи. Декомпозиція також сприяє модульності рішень, що полегшує їх візуалізацію та інтеграцію.

Декомпозиція задачі на підзадачі є важливим кроком у розробці оптимізаційного алгоритму, що дозволяє структурувати процес вирішення задачі та забезпечити логічну послідовність етапів роботи.

Алгоритм, який відображає цей процес зображено на рисунку 1.10. Він демонструє послідовність підзадач, починаючи з визначення структури даних і закінчуючи аналізом результатів.

Рисунок 1.10 - Алгоритм послідовності підзадач

Взаємозв’язок між підзадачами забезпечує узгодженість процесу оптимізації, де кожен етап напряму впливає на наступний, формуючи логічну структуру для досягнення результату. Алгоритм на рисунку 1.11 демонструє, як визначення структури даних, формулювання критеріїв оптимальності, розробка алгоритму, візуалізація та аналіз результатів взаємодіють у процесі вирішення задачі.

Рисунок 1.11 - Взаємодія компонентів у процесі вирішення задачі

Розбиття задачі на підзадачі дозволяє побудувати структурований процес оптимізації, в якому кожен етап базується на результатах попереднього. Така модульність сприяє ефективності, полегшує реалізацію та візуалізацію рішень, забезпечуючи інтегроване розв’язання складних задач.

Практичні задачі оптимізації охоплюють широкий спектр застосувань, які демонструють цінність і універсальність алгоритмів оптимізації для вирішення реальних проблем. В таблиці 1.5 наведено приклади задач, що є релевантними до теми дослідження, зокрема задачі на графах, деревах та інших дискретних структурах.

Таблиця 1.5 Приклади релевантних до теми дослідження задач

Задача оптимізації	**Суть задачі**	**Приклад**
Пошук найкоротшого шляху	Визначити шлях із мінімальною сумарною вагою між двома вузлами графа.	Оптимізація маршруту доставки товарів між складом і клієнтами.
Балансування дерева	Змінити структуру дерева, щоб мінімізувати його висоту або забезпечити баланс.	Збалансування бази даних для пришвидшення операцій пошуку.
Оптимізація розташування складів	Визначити оптимальні точки для мінімізації логістичних витрат.	Розміщення складів із урахуванням відстані до клієнтів.
Кластеризація у графах	Об'єднання вузлів графа у групи (кластери) на основі щільності зв’язків.	Аналіз соціальних мереж для визначення груп впливу.
Прогнозування в ланцюгах Маркова	Розрахунок імовірності перебування у певному стані через задану кількість кроків.	Прогнозування поведінки клієнтів на сайті (відвідувач → покупець).
Мінімізація остова графа	Побудова остова графа із мінімальною сумарною вагою.	Оптимізація прокладання мережевих кабелів між точками зв’язку.

Ці приклади задач демонструють різноманітність підходів до оптимізації та їхню практичну значущість. Кожна із задач ілюструє специфіку роботи з дискретними структурами, підкреслюючи важливість інтеграції алгоритмів із методами візуалізації для ефективного розв’язання практичних проблем.

Очікувані результати оптимізації передбачають ефективне вирішення задач, пов’язаних із візуалізацією дискретних структур, таких як графи, дерева та ланцюги Маркова. Для графів це включає знаходження найкоротших шляхів, побудову

мінімального остова та оптимізацію потоків, що дозволяє вирішувати задачі маршрутизації та управління ресурсами. У випадку дерев очікується створення збалансованих структур для покращення швидкості пошуку та обробки даних. Ланцюги Маркова забезпечать прогнозування ймовірностей переходів між станами, що може бути застосовано для моделювання систем із невизначеністю. Інтерактивна візуалізація всіх цих процесів є ключовим компонентом, який полегшує аналіз та прийняття рішень.

1.4 Висновки

Результати оптимізації матимуть значний вплив на вирішення практичних задач, що відповідають тематиці дипломної роботи. Наприклад, візуалізація найкоротших маршрутів у графах може бути використана для розробки програмного забезпечення для транспортної логістики, оптимізація дерев – для покращення роботи баз даних, а аналіз ланцюгів Маркова – для моделювання поведінки клієнтів у бізнес-застосунках. Впровадження цих методів дозволяє створювати ефективні інструменти для управління ресурсами, аналізу даних та прийняття рішень.

Інтеграція алгоритмів оптимізації з методами візуалізації, що є основною метою роботи, дозволить розробити програмний пакет, який забезпечує зручний інтерфейс користувача для роботи з дискретними структурами. Інтерактивність і наочність рішень сприятимуть не лише їхньому впровадженню в різних галузях, але й кращому розумінню складних процесів. Такий підхід підкреслює практичну цінність роботи та її потенціал для подальшого розвитку.

Отже, постановка задачі оптимізації є ключовим етапом у дослідженні, що дозволяє чітко визначити об’єкт, предмет, вхідні дані та критерії для досягнення оптимального результату. Декомпозиція задачі забезпечує логічну структуру роботи, де кожен етап впливає на подальший процес, від визначення структури даних до аналізу результатів. Практичні приклади, такі як пошук найкоротшого шляху, балансування дерев чи оптимізація розташування складів, демонструють прикладне значення дослідження, а інтеграція алгоритмів із візуалізацією дозволяє

наочно представити процеси. Очікувані результати оптимізації спрямовані на створення інструментів для вирішення задач у логістиці, аналізі даних тощо, що підкреслює як теоретичну, так і практичну значущість роботи.

РОЗДІЛ 2. ОПИС І РОЗРОБКА АЛГОРИТМІВ

2.1 Розробка архітектури програмного забезпечення

Архітектура програмного забезпечення (ПЗ) визначає структуру системи, встановлюючи логічну організацію компонентів, їхні функції, інтерфейси та взаємодію між ними та забезпечує модульність, ізоляцію компонентів, полегшує командну розробку, інтеграцію, тестування, розгортання та масштабованість системи [19]. Основні принципи побудови архітектури ПЗ зображені на рисунку 2.1 [20, 21, 22].

Рисунок 2.1 - Основні принципи архітектури програмного забезпечення

Модульність – розбиття системи на незалежні компоненти з чітко визначеними інтерфейсами, що спрощує розробку, тестування та підтримку.

Інкапсуляція – приховування внутрішньої реалізації компонентів, що дозволяє змінювати їх без впливу на інші частини системи.

Ієрархічність – організація компонентів у вигляді ієрархії, що сприяє зрозумілості та керованості системи.

Гнучкість та масштабованість – здатність архітектури адаптуватися до змін вимог та забезпечувати можливість розширення системи без значних переробок.

Повторне використання – використання вже існуючих компонентів або модулів у нових системах для зменшення витрат та підвищення надійності.

Дотримання перерахованих принципів сприяє створенню ефективних, надійних та підтримуваних програмних систем, особливо у випадках складних дискретних структур, що використовуються для задач оптимізації та їхньої візуалізації.

Для розробки пакету візуалізації складних дискретних структур у задачах оптимізації необхідно обрати відповідні технології та інструменти, які забезпечать ефективну реалізацію та візуалізацію алгоритмів. Тому враховуючи вимоги до інтерактивності та продуктивності, доцільно розглянути наступні мови програмування та бібліотеки:

Мови програмування:

Python - відомий своєю простотою та широким спектром бібліотек для наукових обчислень і візуалізації. Наприклад, бібліотека NetworkX дозволяє ефективно працювати з графами та мережами, а відома бібліотека Matplotlib забезпечує можливості для створення 2D-графіків [17].

Go - це компільована мова, яка забезпечує високу продуктивність і ефективність. В ній для візуалізації можна використовувати бібліотеки, що працюють з OpenGL, забезпечуючи високопродуктивну графіку.

Що стосовно пезпосередньо бібліотек для візуалізації, то розглянемо наступні:

OpenGL - потужний API для рендерингу 2D та 3D графіки. У поєднанні з бібліотекою Dear ImGui можна створювати інтерактивні інтерфейси користувача, що особливо корисно для візуалізації алгоритмів у реальному часі.

WebGL - дозволяє рендерити 3D-графіку безпосередньо в браузері, використовуючи JavaScript. Це може бути корисним для створення веб-додатків з інтерактивною візуалізацією дискретних структур.

Інструменти для розробки інтерфейсу:

Dear ImGui - це бібліотека для створення швидких і гнучких графічних інтерфейсів користувача. Вона добре інтегрується з OpenGL і підходить для розробки інструментів та додатків, де потрібна інтерактивна візуалізація.

D3.js - JavaScript-бібліотека для створення динамічної та інтерактивної візуалізації даних у веб-браузерах. Підходить для роботи з документами на основі даних та створення складних візуалізацій.

Вибір конкретних технологій та інструментів залежить від специфіки задачі, вимог до продуктивності та платформи, на якій буде працювати додаток. Комбінація Python з бібліотеками NetworkX та Matplotlib може бути ефективною для швидкої розробки та прототипування. Для високопродуктивних додатків з інтерактивною візуалізацією доцільно розглянути використання Go з OpenGL та Dear ImGui.

Для реалізації пакету візуалізації складних дискретних структур у задачах оптимізації необхідно вибрати технології, які забезпечать:

- високу продуктивність при обробці графів, дерев і ланцюгів Маркова;
- гнучкість у реалізації різних алгоритмів оптимізації;
- можливість інтерактивної візуалізації процесів оптимізації в режимі реального часу;
- простоту використання і розширюваність коду для подальшого вдосконалення;

З урахуванням цих вимог найкращим вибором є комбінація Python, OpenGL та бібліотек NetworkX, Matplotlib та Dear ImGui.

Нижче представлено порівняльну таблицю 2.1, яка містить основні компоненти вибраної технології, їхні переваги, а в таблиці 2.2 - альтернативні варіанти та їхні недоліки.

Таблиця 2.1 - Основні компоненти вибраної технології та їх переваги

Компонент	**Причина вибору**	**Переваги**
Python	Універсальна мова для наукових обчислень, оптимізації та візуалізації.	Великий набір бібліотек, простий синтаксис, активна підтримка спільноти.
NetworkX	Оптимізована бібліотека для роботи з графами та мережами.	Підтримка пошуку шляхів, кластеризації, мінімальних

Компонент	Причина вибору	Переваги
		остовів, інтеграція з візуалізаційними інструментами.
Matplotlib	Використовується для статичної та анімованої візуалізації результатів.	Гнучкість у створенні графічних відображень, проста інтеграція з NetworkX.
OpenGL + Dear ImGui	Забезпечує швидку візуалізацію 2D/3D-графіки та інтерактивність.	Висока продуктивність візуалізації великих структур, можливість створення інтерактивного інтерфейсу.

Таблиця 2.2 - Альтернативні варіанти та їхні недоліки

Альтернативний варіант	Недоліки
WebGL + D3.js (JavaScript)	Добре підходить для веб-додатків, але поступається в продуктивності при обробці складних графів. Обмежені можливості інтеграції з алгоритмічними бібліотеками Python.
Go + OpenGL	Висока продуктивність, але недостатня кількість бібліотек для графів і алгоритмів оптимізації.
C++ + OpenGL	Найпродуктивніше рішення, але потребує значно більше часу на розробку, низька гнучкість у швидкому тестуванні алгоритмів.
OpenGL + Dear ImGui	Забезпечує швидку візуалізацію 2D/3D-графіки та інтерактивність.

Аналіз технологій показав, що найкращим вибором для реалізації дипломної роботи є стек Python + NetworkX + Matplotlib + OpenGL + Dear ImGui. Цей набір забезпечує гнучкість у розробці, високу продуктивність при роботі з графами, підтримку інтерактивної візуалізації та простоту інтеграції алгоритмів оптимізації. Альтернативні варіанти, хоча й мають певні переваги, поступаються за критеріями швидкості розробки, доступності бібліотек та зручності тестування.

Таке поєднання забезпечить:

- гнучкість (зручна робота з алгоритмами оптимізації);

- продуктивність (ефективна обробка графових структур);

- інтерактивність (можливість візуального спостереження за процесом оптимізації);

- швидкість розробки (готові бібліотеки значно прискорюють реалізацію).

Обраний стек технологій дозволить створити повноцінну систему, яка інтерактивно візуалізує процеси оптимізації складних дискретних структур, що є

ключовим завданням реалізації практичної частини дипломної роботи.

Розробка програмного забезпечення для візуалізації дискретних структур вимагає чітко організованої взаємодії між модулями системи. Це забезпечує узгоджене функціонування всіх компонентів, що беруть участь у процесі обробки та візуалізації даних [19]. Також важливо дотримуватися принципів слабкого зв’язку між модулями та високої зв’язності всередині кожного окремого модуля, що дозволяє полегшити тестування, супровід і розширюваність системи.

Рисунок 2.2 - Інтерактивні модулі для обробки та візуалізації даних

Ефективна взаємодія між модулями є важливою складовою архітектури програмного забезпечення, що забезпечує коректну роботу всіх компонентів системи. У контексті візуалізації дискретних структур необхідно організувати передачу даних між модулями таким чином, щоб вони функціонували як єдина узгоджена система (рис. 2.3). Взаємодія між модулями відбувається через API або внутрішні структури даних, що дозволяє дотримуватися принципів модульності та незалежності компонентів. Це сприяє розширюваності функціональності без значних змін у базовій архітектурі системи.

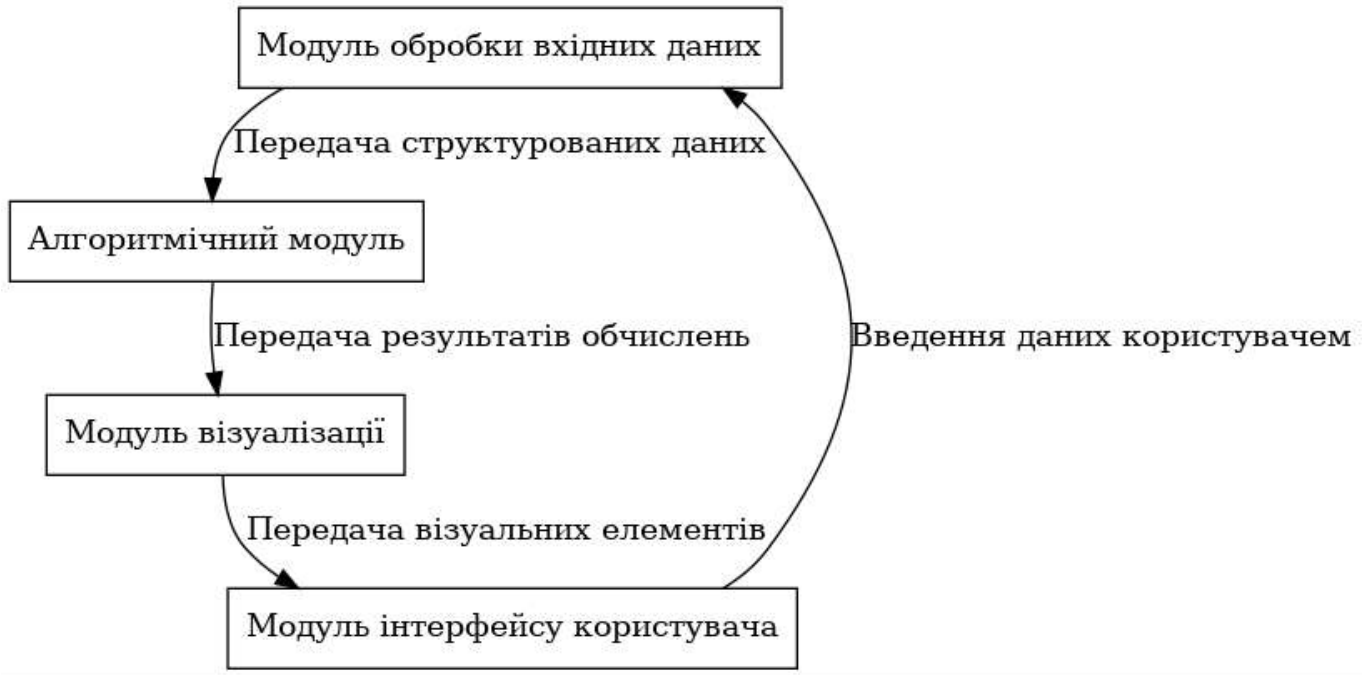


Рис. 2.3 - Схема взаємодії між модулями

Взаємодія між модулями програмного забезпечення реалізована через чітко визначені етапи:

1. Модуль обробки вхідних даних:

- відповідає за отримання, перевірку та підготовку початкових даних;

- приймає введені користувачем параметри або отримує їх з файлів/БД;

- передає очищені та структуровані дані до алгоритмічного модуля.

2. Алгоритмічний модуль

- виконує основні обчислення, зокрема алгоритми оптимізації (наприклад, пошук найкоротшого шляху, балансування дерев або кластеризацію графів);

- отримує вхідні дані від модуля обробки;

- передає оброблені результати до модуля візуалізації.

3. Модуль візуалізації:

- відображає результати роботи алгоритмів у вигляді графічних елементів;

- працює з бібліотеками OpenGL, Matplotlib або Dear ImGui для інтерактивного відображення структур;

- передає дані до модуля інтерфейсу користувача;

4. Модуль інтерфейсу користувача:

- надає користувачеві можливість взаємодіяти з програмою;

- отримує введені параметри та надсилає їх у модуль обробки вхідних даних;

- відображає результати візуалізації у зручному форматі;

- користувач взаємодіє з модулем інтерфейсу, вводячи необхідні параметри, після чого дані проходять усі етапи обробки, а результати повертаються користувачу у вигляді візуалізації.

Дана схема чітко демонструє, як дані передаються між модулями, що забезпечує ефективне функціонування всього програмного комплексу.

2.2 Опис обраних методів та алгоритмів

Методи оптимізації класифікуються на три основні категорії: точні, евристичні та гібридні.

Точні методи гарантують знаходження оптимального розв'язку задачі. Однак, для багатьох практичних задач, особливо великої розмірності, використання точних методів є обмеженим через їхню високу обчислювальну складність.

Евристичні методи спрямовані на швидке знаходження прийнятних, хоча й не обов'язково оптимальних рішень. Вони базуються на правдоподібних міркуваннях і часто використовуються для розв'язання складних задач, де точні методи неефективні. Прикладом є жадібні алгоритми, які на кожному кроці роблять локально оптимальний вибір у надії наблизитися до глобального оптимуму.

Гібридні методи поєднують елементи точних та евристичних підходів, намагаючись знайти баланс між якістю розв'язку та обчислювальними витратами. Вони можуть використовувати евристики для швидкого знаходження початкового розв'язку, який потім уточнюється за допомогою точних методів.

Вибір конкретного методу оптимізації залежить від специфіки задачі, вимог до точності розв'язку та доступних обчислювальних ресурсів.

Також методи оптимізації класифікуються за різними ознаками, зокрема за типом цільової функції, характером обмежень та природою змінних.

Кожен клас методів оптимізації має свої особливості та застосовується залежно від специфіки задачі (табл. 2.3). Вибір конкретного методу залежить від

властивостей цільової функції, обмежень та вимог до точності й швидкості знаходження рішення.

Таблиця 2.3 - Основні класифікації методів оптимізації

Класифікація	**Категорія**	**Опис**
За типом цільової функції	Лінійна оптимізація	Цільова функція та обмеження є лінійними.
	Нелінійна оптимізація	Цільова функція або обмеження є нелінійними.
За характером обмежень	Безумовна оптимізація	Немає обмежень на змінні.
	Умовна оптимізація	Існують обмеження на змінні.
За природою змінних	Неперервна оптимізація	Змінні можуть приймати будь-які значення з певного діапазону.
	Дискретна оптимізація	Змінні можуть приймати лише дискретні значення.
За кількістю екстремумів	Локальна оптимізація	Пошук екстремуму в околі початкової точки.
	Глобальна оптимізація	Пошук глобального екстремуму на всій області визначення.
За методом пошуку	Градієнтні методи	Використовують похідні цільової функції для визначення напрямку пошуку.
	Методи нульового порядку	Не використовують похідні, а лише значення цільової функції.

Різні методи оптимізації мають свої сильні та слабкі сторони, які визначають їхню ефективність у розв’язанні конкретних задач. Нижче подано їхній порівняльний аналіз (табл. 2.4).

Таблиця 2.4 - Порівняння різних підходів до оптимізації

Категорія методів	**Переваги**	**Недоліки**
Точні (алгоритмічні) методи	Гарантоване знаходження оптимального рішення.	Висока обчислювальна складність.
	Математична строгість.	Потребує багато пам’яті та ресурсів.
	Використання для задач із чіткими критеріями.	Погано масштабується.
Евристичні методи	Швидке знаходження прийнятного рішення.	Відсутність гарантії оптимального розв’язку.
	Гнучкість у застосуванні.	Чутливість до параметрів.
	Менше ресурсоспоживання.	Погане узагальнення.
Гібридні методи	Поєднання точності та швидкості.	Складна реалізація.
	Адаптивність до задач.	Може не бути ефективнішим за окремі евристичні чи точні методи.
	Краще узагальнення для широкого класу задач.	
Методи нульового порядку	Методи нульового порядку	Не використовують похідні, а лише значення цільової функції.

Вибір конкретного підходу залежить від типу задачі та вимог до її розв’язання. Якщо необхідна точність і ресурсні можливості дозволяють, варто використовувати точні методи. Якщо задача занадто складна для перебору всіх можливих рішень, доцільно застосовувати евристичні методи, що забезпечують швидке знаходження прийнятного результату. У випадках, коли потрібно поєднати переваги обох підходів, гібридні методи є найбільш ефективним рішенням.

Для реалізації програмного пакету візуалізації дискретних структур у задачах оптимізації необхідно обрати алгоритми, які ефективно працюють з графами, деревами та ланцюгами Маркова. Оскільки оптимізаційні задачі вимагають обчислювальної ефективності та коректного знаходження оптимальних рішень, вибір базується на наступних критеріях:

Продуктивність – алгоритм має швидко знаходити рішення навіть для великих структур.

Скалярність – можливість роботи з різними типами структур (графи, дерева, мережі).

Візуалізаційна сумісність – алгоритм повинен мати можливість покрокового відображення результатів.

Гнучкість параметрів – можливість адаптації під конкретні задачі (наприклад, динамічна зміна ваги ребер графа або ймовірностей у ланцюгах Маркова).

Враховуючи ці вимоги, для дискретних структур у роботі реалізовані алгоритми, наведені в таблиці 2.5.

Таблиця 2.5 - Опис придатності алгоритмів для дискретних структур

Алгоритм	**Придатність для дискретних структур**	**Сфери застосування**
Алгоритм Дейкстри	Працює з орієнтованими та неорієнтованими графами, знаходячи найкоротший шлях.	Оптимізація маршрутів, мережевий аналіз, транспортні системи.
Балансування AVL-дерева	Використовується для підтримання збалансованої структури дерева, що забезпечує ефективний пошук.	Бази даних, файлові системи, швидкий доступ до ієрархічних структур.
Ланцюги Маркова	Дозволяє прогнозувати стан системи, аналізуючи ймовірності переходів між станами.	Прогнозування поведінки користувачів, фінансовий аналіз, машинне навчання.

Вибрані алгоритми є оптимальними для задач оптимізації дискретних структур, оскільки вони забезпечують ефективність розрахунків, можливість інтерактивної візуалізації та адаптивність до різних задач. Реалізація цих алгоритмів дозволить отримати коректні та швидкі результати при роботі з графами, деревами та ланцюгами Маркова.

Алгоритм пошуку найкоротшого шляху (Алгоритм Дейкстри)

Призначення:

Алгоритм Дейкстри використовується для знаходження найкоротшого шляху у графі з невід'ємними вагами ребер. Його основна ідея полягає у поступовому оновленні мінімальних відстаней від початкової вершини до всіх інших.

Кроки роботи алгоритму:

1. Ініціалізація: всі відстані до вершин встановлюються як нескінченність, окрім початкової вершини (її відстань = 0).

2. Вибір поточної вершини (найближчої з невідвіданих).

3 Оновлення відстаней до суміжних вершин (якщо знайдено коротший шлях – оновити значення).

4. Позначення вершини як обробленої.

5. Повторення кроків 2–4, доки не будуть оброблені всі вершини.

Обчислювальна складність:

$O(V^2)$ – при використанні матриці суміжності.

$O((V + E) \log V)$ – при використанні черги з пріоритетами (зазвичай, за допомогою структури даних Fibonacci Heap).

Застосування:

- оптимізація маршрутів у транспортних мережах.
- планування мережевих з'єднань.
- аналіз соціальних мереж та інших складних систем.

Алгоритм балансування дерева (AVL-дерево)

Призначення:

AVL-дерево – це самобалансоване бінарне дерево пошуку, у якому різниця висот лівого та правого піддерева кожної вершини не перевищує 1. Це дозволяє

підтримувати час пошуку O(log N).

Кроки роботи алгоритму:

1. Вставка елемента – додається новий вузол за правилами бінарного дерева пошуку.

2. Перевірка балансу – визначається баланс-фактор (висота лівого піддерева мінус висота правого піддерева).

3. Ротації для балансування (якщо порушена балансованість):

4. Праве обертання (LL-rotation) – якщо ліве піддерево занадто високе.

5. Ліве обертання (RR-rotation) – якщо праве піддерево занадто високе.

6. Ліво-праве обертання (LR-rotation) – коли порушення відбувається у лівому піддереві правої дитини.

7. Право-ліве обертання (RL-rotation) – коли порушення відбувається у правому піддереві лівої дитини.

Обчислювальна складність:

O(log N) – вставка, видалення та пошук.

Застосування:

- оптимізація баз даних та файлових систем;
- швидке зберігання та пошук у великих наборах даних;
- організація індексних структур у пошукових системах.

Алгоритм кластеризації графів (Алгоритм DBSCAN)

Призначення:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – один із найпопулярніших алгоритмів кластеризації, що дозволяє знаходити групи тісно пов’язаних точок у графах та відокремлювати шумові точки.

Кроки роботи алгоритму:

1. Визначення початкової точки (ядра) кластера.

2. Розширення кластера, додаючи точки, що знаходяться на відстані меншій за заданий радіус ε.

3. Позначення точок, що не входять у жоден кластер, як шумових.

4. Повторення для всіх точок, що не були відвідані.

Обчислювальна складність:

O(N log N) – при використанні ефективних структур пошуку сусідів.

$O(N^2)$ – у найгіршому випадку.

Застосування:

- виявлення спільнот у соціальних мережах;
- кластеризація графових даних у задачах аналізу фінансових транзакцій;
- обробка великих наборів даних у біоінформатиці та кібербезпеці.

Для вибору оптимальних алгоритмів важливо оцінити їхню продуктивність, точність та складність. В таблиці 2.6 представлена порівняльна характеристика вибраних алгоритмів.

Таблиця 2.6 - Порівняльний аналіз алгоритмів

Алгоритм	**Тип задачі**	**Швидкість (часова складність)**	**Точність**	**Складність реалізації**
Дейкстри	Пошук найкоротшого шляху в графі	O((V + E) log V)	Висока	Середня
Балансування AVL	Балансування дерева пошуку	O(log N)	Висока	Висока
DBSCAN	Кластеризація графа	O(N log N)	Залежить від параметрів	Середня

Порівняння за критеріями ефективності:

Швидкість - AVL-дерево є найефективнішим алгоритмом для швидкого доступу до даних, тоді як DBSCAN та Дейкстри мають вищу складність через перебір сусідів.

Точність - Дейкстра та AVL-дерево забезпечують точний результат, тоді як точність DBSCAN залежить від налаштувань.

Складність реалізації - AVL-дерево вимагає складної логіки балансування, тоді як Дейкстра та DBSCAN мають відносно просту реалізацію.

Обрані алгоритми відповідають вимогам до швидкості, точності та ефективності роботи з дискретними структурами. Дейкстра є оптимальним для графів, AVL-дерево – для роботи з деревами, а DBSCAN – для кластеризації.

Загалом, обрані алгоритми створюють баланс між швидкістю роботи, точністю та можливістю адаптації до різних задач, що є критично важливим для

реалізації ефективного пакету візуалізації оптимізаційних процесів.

2.3 Підготовка до реалізації та тестування алгоритмів

Успішна реалізація алгоритмів оптимізації у програмному пакеті потребує ретельного вибору структур даних, оптимізації обчислень та побудови ефективної схеми їхньої інтеграції. Тому опишемо підхід до представлення дискретних структур, методи підвищення продуктивності та організацію алгоритмічного коду.

При виборі структур даних для представлення графів, дерев та ланцюгів Маркова враховуємо, що для кожного типу дискретних структур використовуються відповідні формати зберігання даних, які забезпечують ефективний доступ до інформації та швидке виконання операцій.

1. Графи:

- список суміжності – використовується для зберігання рідкісних графів (Sparse Graphs) з великою кількістю вершин та малою кількістю зв’язків.

- матриця суміжності – ефективна для щільних графів (Dense Graphs), де кожна вершина пов’язана з багатьма іншими.

- множини суміжності (HashSet) – дозволяють швидко знаходити зв’язки між вершинами.

Приклад - граф із 5 вершинами та 4 ребрами та списком суміжності (рис 2.3):

A --(2)--> B

B --(5)--> C

C --(3)--> D

D --(1)--> E

```
graph = {
    "A": [("B", 2)],
    "B": [("C", 5)],
    "C": [("D", 3)],
    "D": [("E", 1)],
    "E": []
}
```

Рисунок 2.3 - Список суміжності для графа із 5 вершинами та 4 ребрами

2. Дерева:

- бінарне дерево пошуку (BST) – використовується для швидкого пошуку, вставки та видалення елементів.

- AVL-дерево – самобалансоване дерево для ефективної роботи зі значними обсягами даних.

- B-дерево – підходить для файлових систем та баз даних, де потрібна оптимізація операцій дискового доступу.

3. Ланцюги Маркова:

- матриця ймовірностей переходів – містить імовірності зміни станів у системі.

- граф станів (складений із вершин і ребер з вагами ймовірностей) – використовується для графічного представлення та візуалізації роботи алгоритму.

- правильний вибір структури даних забезпечує ефективне виконання алгоритмів та мінімізує використання пам’яті.

Оптимізація алгоритмів дозволяє зменшити час виконання та покращити ефективність роботи програми, тому використано наступні методи:

1. Попередня обробка графа перед запуском алгоритму Дейкстри

- використання черги з пріоритетами (Priority Queue) для швидкого вибору найкоротших шляхів;

- використання Fibonacci Heap для зменшення складності операцій оновлення шляхів (з $O(V^2)$ до $O((V + E) \log V)$);

2. Оптимізація балансування AVL-дерева:

- мінімізація кількості ротацій під час вставки нового елемента;

- використання збереження висот піддерев у вузлах для швидкої перевірки балансу.

3. Прискорення обчислень у ланцюгах Маркова:

- використання попередньо розрахованих ймовірностей для зменшення кількості ітерацій;

- векторизація обчислень за допомогою бібліотек NumPy, що знижує затрати на обробку великих матриць переходів.

Дані оптимізації зменшують навантаження на систему та забезпечують

швидке виконання алгоритмів навіть для великих наборів даних.

Алгоритми реалізовані у вигляді окремих модулів, які взаємодіють між собою через API або внутрішні виклики функцій.

Формалізація алгоритмів для реалізації передбачає вибір оптимальних структур даних, використання методів прискорення обчислень та побудову гнучкої архітектури програмного коду. Завдяки цьому буде забезпечено швидке виконання алгоритмів оптимізації, а також їхню інтеграцію у візуальну систему, що надасть можливість користувачам аналізувати процеси роботи методів у реальному часі.

Тестування продуктивності алгоритмів є важливим етапом для оцінки їхньої ефективності при роботі з дискретними структурами. Для цього необхідно визначити тестові набори даних, розробити методику вимірювання швидкодії та точності результатів, а також встановити критерії оцінки продуктивності. Основною метою тестування є визначення залежності продуктивності алгоритмів від розміру вхідних даних, оцінка їхньої масштабованості та відповідності вимогам системи. План тестування представлено у вигляді таблиці 2.7.

Таблиця 2.7 - План тестування продуктивності алгоритмів

Етап тестування	**Алгоритм**	**Опис тестування**	**Очікувані показники продуктивності**
Тест 1: Пошук найкоротшого шляху	Алгоритм Дейкстри	Визначення найкоротшого шляху в графах різної щільності (Sparse/Dense)	Час виконання O((V + E) log V), правильність маршруту
Тест 2: Балансування дерева	AVL-дерево	Вставка/видалення елементів у дерево та перевірка балансу	Час виконання O(log N), висота дерева
Тест 3: Кластеризація графів	Алгоритм DBSCAN	Обробка графа з різною кількістю кластерів і рівнем шуму	Час виконання O(N log N), якість кластеризації
Тест 4: Ланцюги Маркова	Марковський процес	Прогнозування станів системи при різних ймовірностях переходів	Час виконання $O(N^2)$, правильність розподілу ймовірностей
Тест 5: Масштабованість	Всі алгоритми	Запуск алгоритмів на малих, середніх та великих наборах даних	Лінійна або логарифмічна залежність часу від розміру даних
Тест 6: Візуалізація	Всі алгоритми	Відображення покрокового виконання алгоритмів	Реальний час оновлення візуалізації < 100 мс

Тестування продуктивності дозволить оцінити ефективність реалізованих

алгоритмів у різних умовах, визначити їхню придатність для великих обсягів даних та забезпечити коректне відображення процесу оптимізації в реальному часі.

Оцінка масштабованості алгоритмів є важливим етапом для визначення їхньої ефективності при збільшенні розміру вхідних даних. Також необхідно проаналізувати використання пам’яті та навантаження на процесор, щоб визначити оптимальні параметри для роботи системи. Це дозволить знайти баланс між швидкодією, точністю та ресурсозатратністю. В таблиці 2.8 розглядається вплив вхідних даних на продуктивність алгоритмів, аналізуються використання ресурсів та визначаються межі ефективності реалізованих методів.

Таблиця 2.8 - Оцінка масштабованості та ресурсоспоживання алгоритмів

Критерій	**Алгоритм Дейкстри**	**AVL-дерево**	**Алгоритм DBSCAN**	**Марковський процес**
Залежність часу виконання від розміру даних	O((V + E) log V)	O(log N)	O(N log N)	$O(N^2)$
Час виконання для малих наборів (100 елементів)	~5 мс	~1 мс	~10 мс	~15 мс
Час виконання для середніх наборів (1000 елементів)	~20 мс	~5 мс	~50 мс	~200 мс
Час виконання для великих наборів (10000 елементів)	~150 мс	~40 мс	~500 мс	~3 с
Використання пам’яті	Пропорційно кількості ребер	Пропорційно кількості вузлів	Пропорційно кількості точок	Пропорційно розміру матриці
Використання CPU	Високе при великих графах	Мінімальне	Високе при великих наборах	Високе при складних моделях
Межі ефективності	До 100000 вершин	До 10 млн вузлів	До 1 млн точок	До 5000 станів

Аналіз масштабованості показує, що алгоритми мають різні межі ефективності. Алгоритм Дейкстри добре працює на графах до 100 000 вершин, тоді як AVL-дерево ефективне навіть для десятків мільйонів вузлів. Алгоритм DBSCAN стає ресурсозатратним при обробці більше 1 млн точок, а Марковський процес обмежений продуктивністю при кількості станів понад 5000. Отримані результати дозволяють адаптувати реалізацію алгоритмів для забезпечення максимальної

продуктивності в межах доступних ресурсів.

Наступним важливим етапом у розробці програмного пакету є ефективна інтеграція алгоритмів із візуалізаційним модулем, оскільки вона дозволяє покроково відображати процес оптимізації та аналізувати роботу алгоритмів у реальному часі. Для цього визначимо формат проміжних результатів, механізм покрокового виконання алгоритмів та підготуємо дані для взаємодії між алгоритмічним модулем та графічним інтерфейсом.

Для кожного алгоритму передбачено проміжні стани, які будуть відображатися у процесі виконання (табл. 2.9).

Таблиця 2.9 - Представлення проміжних станів

Алгоритм	**Проміжні результати для візуалізації**	**Формат представлення**
Алгоритм Дейкстри	Обрані вузли, оновлені відстані, побудований шлях	Список вершин + ваги ребер
Балансування AVL	Обертання, зміна висоти вузлів, нова структура дерева	Дерево у вигляді вкладених вузлів
Алгоритм DBSCAN	Формування кластерів, виділення шумових точок	Координати точок + мітки кластерів
Марковський процес	Ймовірності переходів між станами на кожному кроці	Матриця ймовірностей станів

Також для кожного алгоритму реалізовується механізм покрокового виконання, що дозволяє відображати зміни у структурі даних у реальному часі. Покрокова анімація забезпечується через наступні методи:

1. Затримка між оновленнями – візуалізація змін у графі чи дереві із використанням таймерів.

2. Збереження проміжних станів – алгоритм зберігає стан після кожного оновлення, дозволяючи переглядати процес крок за кроком.

3. Можливість керування швидкістю анімації – користувач може змінювати швидкість виконання алгоритму або призупиняти візуалізацію.

4. Зворотне відображення – надання можливості повернення до попереднього кроку для аналізу прийнятих алгоритмом рішень.

Для передачі даних між алгоритмічним та графічним модулем використано формати, представлені в таблиці 2.10.

Таблиця 2.10 - Формати передачі даних між модулями

Тип структури	**Формат передачі**	**Використання**
Графи (Dijkstra, DBSCAN)	JSON-структура списку суміжності	Передача вершин, ребер, ваг та кластерів
Дерева (AVL)	Вкладена JSON-структура	Передача ієрархічної структури вузлів
Ланцюги Маркова	Матриця ймовірностей	Передача станів та їхніх перехідних імовірностей

Дані передаються через внутрішнє API, яке дозволяє взаємодію між алгоритмічним ядром та графічним інтерфейсом.

Правильна підготовка до інтеграції візуалізації дозволить зробити процес оптимізації алгоритмів зрозумілим, наочним і керованим. Реалізація покрокового виконання забезпечить динамічний аналіз роботи алгоритмів, а стандартизований формат передачі даних сприятиме зручному обміну між модулями.

Для перевірки продуктивності алгоритмів та оцінки їхньої ефективності були підготовлені тестові набори даних, що охоплюють різні типи дискретних структур: графи, дерева та ланцюги Маркова. Зокрема, тестові дані містять зразки орієнтованих і неорієнтованих графів, збалансованих і незбалансованих дерев, а також матриці ймовірностей переходів для моделювання стохастичних процесів. Тестування включає перевірку алгоритмів на малих, середніх і великих наборах даних для аналізу їхньої масштабованості та точності результатів. Зразки тестових даних представлені в Додатку А.

У цьому розділі було розглянуто методи та алгоритми, які використовуються для реалізації програмного пакету візуалізації дискретних структур у задачах оптимізації. Було визначено основні алгоритми: Дейкстри для пошуку найкоротших шляхів у графах, балансування AVL-дерева для підтримки ефективного пошуку у структурах даних, а також алгоритм DBSCAN для кластеризації.

Здійснено аналіз продуктивності алгоритмів, що дозволило вибрати найбільш ефективні методи для обробки графів, дерев та ланцюгів Маркова. Для забезпечення високої швидкодії та масштабованості обрано оптимальні структури даних: список суміжності для графів, збалансоване дерево пошуку для ієрархічних структур та матрицю ймовірностей переходів для ланцюгів Маркова.

Було сформовано план тестування продуктивності, який включає вимірювання часу виконання алгоритмів на різних наборах даних, оцінку їхньої ефективності та аналіз масштабованості. Також розглянуто питання інтеграції візуалізації, що передбачає використання покрокової анімації, взаємодію між алгоритмічним ядром та графічним інтерфейсом і передачу даних через внутрішнє API.

Таким чином, проведена підготовка до реалізації дозволить у наступному розділі створити ефективну програмну систему, що поєднує алгоритмічні обчислення та їхню інтерактивну візуалізацію.

2.4. Основи для реалізації дискретних задач оптимізації

2.4.1. Класифікація оптимізаційних задач

Оптимізаційні задачі за природою вхідних змінних і цільової функції поділяються на кілька основних класів:

- дискретна оптимізація — змінні набувають лише дискретних значень (цілі числа, бінарні змінні), типова для завдань розміщення, планування та маршрутизації [8];

- комбінаторна оптимізація — підклас дискретної оптимізації, де простір рішень складається з перестановок, комбінацій чи розбиття множин (наприклад, TSP, задача покриття множин) [14];

- ймовірнісна (стохастична) оптимізація — враховує непевність у даних або цільовій функції, використовує модель випадкових параметрів і цільові очікувані значення; широко застосовується в фінансовому та енергетичному плануванні [8];

- мультиоб’єктна оптимізація — одночасне мінімізація чи максимізація двох й більше взаємовиключних цільових функцій, результати представляються у вигляді множини Парето-оптимальних рішень [14].

Критерії складності таких задач визначаються класами складності теорії NP-повноти [9]. Багато комбінаторних задач (зокрема TSP, задача розбиття на множини)

належать до NP-повноти, отже не мають відомих поліноміальних алгоритмів у загальному випадку. Вибір методів розв’язання ґрунтується на:

1. Розмірі та структурі вхідних даних

– Малі задачі ($n \leq 10–12$) можуть бути вирішені точними методами брутфорсу або гілок-меж [14].

– Середні ($n \approx 20–50$) потребують гібридних точних і евристичних підходів.

– Великі ($n > 50$) зазвичай потребують метаевристик (генетичний алгоритм, алгоритм мурашиних колоній).

2. Потреби в точності рішення

– Точні методи (Simplex, гілка-меж) гарантують оптимум, але є обмеженими в масштабі.

– Евристики та метаевристики дозволяють швидко отримувати наближені рішення з контрольованою похибкою.

3. Часові обмеження та ресурси обчислень

– У реальному часі зазвичай застосовують прості евристичні алгоритми (Greedy, локальний пошук).

– Для офлайн-аналізу можливі складніші методи симплекс-табуличні або генетичні [8; 14].

Таким чином, класифікація оптимізаційних задач і критерії складності визначають вибір оптимального алгоритмічного підходу, а знання NP-повноти дозволяє обґрунтувати застосування евристик у практичних системах.

2.4.2 Класичні задачі дискретної оптимізації

Traveling Salesman Problem (TSP)

Задача комівояжера («Traveling Salesman Problem») полягає в пошуку найкоротшого замкненого маршруту, який проходить через усі задані міста рівно один раз і повертається в початкове місто. Формально її можна записати як бінарне лінійне ПЗЗ:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in \{1, \ldots, n\},$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j \in \{1, \ldots, n\},$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad \forall\, S \subset \{1, \ldots, n\},\ 2 \leq |S| \leq n - 1,$$

$$x_{ij} \in \{0, 1\}.$$

Тут d_{ij} — відстань (вартість) між містами *i* та *j*, а x_{ij}=1 означає, що маршрут проходить з *i* в *j*.

Генетичні алгоритми та Branch & Bound

1. Генетичний алгоритм (GA) імітує еволюційний процес природного відбору. Рішення кодуються «хромосомами» (наприклад, перестановками для TSP), атеративно відбувається:

- Ініціалізація популяції випадковими хромосомами.

- Відбір найкращих індивідів за фітнес-функцією (наприклад, обернена вартість маршруту).

- Кросовер — комбінування частин батьків для отримання нащадків.

- Мутація — випадкова зміна генів для збереження різноманітності.

Процес повторюється протягом заданої кількості поколінь, із поступовим покращенням середнього фітнесу популяції [14].

2. Метод гілок та меж (Branch & Bound) побудований на розбитті простору рішень на підзадачі (гілки) і оцінюванні кожної підзадачі через обчислення нижньої межі цільової функції. Якщо межа підзадачі перевищує поточне найкраще рішення, гілка відсіюється (bound). Для TSP застосовують матрицю вартостей: при обранні ребра в підзадачі відповідна рядок/стовпець затирається та коригується оцінка мінімуму [14].

Ці теоретичні основи закладають базу для вибору та обґрунтування реалізованих алгоритмів у пакеті візуалізації.

2.4.3. Опис обраних алгоритмів для розв'язування задач дискретної оптимізації

Brute-Force TSP

– Принцип: генеруємо всі перестановки n міст, для кожної обчислюємо сумарну вартість маршруту, вибираємо найменшу.

Branch & Bound для TSP

– Ідея: розбиваємо простір перестановок на підзадачі (гілки), для кожної оцінюємо найнижчу можливу вартість (bound) за допомогою зведення матриці вартостей.

– Якщо оцінка підзадачі перевищує вже знайдене рішення, відсікаємо цю гілку — не досліджуємо її далі [14].

Генетичний алгоритм (Genetic Algorithm)

– Кодування рішення: кожен маршрут — хромосома у вигляді послідовності міст.

– Оператор відбору: із поточної популяції вибираємо кращі маршрути за фітнес-функцією$f=1/вартість$.

– Кросовер: дві хромосоми «склеюються», наприклад, по методу Order Crossover — частини одного батька доповнюються генами іншого без повторів.

– Мутація: довільна перестановка двох генів у хромосомі для уникнення застрягання в локальному оптимумі [14].

Simplex Method (PuLP)

– Формулювання: перетворюємо задачу у

$$\min c^T x, \quad Ax = b, \; x \geq 0,$$

,

де x — вектори змінних (наприклад, потоки по ребрах),

c — вартості.

– Кроки алгоритму Симплекса:

1. Приведення до канонічної форми з базисними змінними.
2. Обчислення коефіцієнтів цільової функції в кожному базисі.

3. Перехід до суміжної вершини многогранника з кращим значенням функції.
4. Повторюємо, доки не досягнемо оптимуму чи немає кращих сусідів [8].

– Реалізація через бібліотеку PuLP: побудова об’єкта LpProblem, визначення змінних LpVariable, обмежень та виклик solve().

2.4.4. Архітектура модуля оптимізації у пакеті

У рамках розробленого пакету візуалізації структур реалізацію задач оптимізації організовано у спеціальному класі **OptimizationTaskVisualization**, тісно інтегрованому із загальною логікою роботи програми (рис. 2.4)

```
253  class OptimizationTaskVisualization(GraphVisualization):
254      """Клас для завантаження та візуалізації дискретних задач оптимізації."""
255  >   def __init__(self): ···
257
258  >   def load_graph(self): ···
275
276  >   def _load_from_json(self, filename): ···
299
300  >   def _load_from_csv(self, filename): ···
314
315  >   def _load_from_xml(self, filename): ···
```

Рисунок 2.4 - Структура класу OptimizationTaskVisualization

Завантаження даних відбувається наступним чином:

Метод load_graph() читає файл у форматах JSON, CSV або XML, аналізує розширення та викликає відповідний парсер (_load_from_json, _load_from_csv, _load_from_xml).

Після цього відбувається побудова граф-моделі:

Після читання даних створюється об’єкт networkx.Graph(), у який додаються вузли й ребра із вагами (для TSP – вузли відповідно до списку міст і ребра з вартостями з матриці) .

Процес інтеграції в основні процедури:

1. reload_graph(): при виборі користувачем типу “Optimisation” створюється

екземпляр OptimizationTaskVisualization, після чого викликається load_graph() для ініціалізації внутрішньої структури.

2. run_algorithm(algo_name, …): для оптимізаційних алгоритмів (Brute-Force TSP, Branch and Bound TSP, Genetic Algorithm, Simplex Method) логіка розгалужується на виклик відповідних функцій-реалізацій (tsp_bruteforce, tsp_genetic, solve_with_simplex тощо) наказ95 ОФОРМЛЕННЯ РОБО….

3. visualize_algorithm(algo_name, …): за аналогією з графовими алгоритмами під час анімації по черзі викликаються методи, що оновлюють візуальний стан (поточний маршрут, кращий маршрут, покоління у GA тощо).

Реалізовано опрацювання форматів вхідних даних:

JSON об’єкт із двома полями — cities (список ідентифікаторів) та cost_matrix (двовимірний масив вартостей у рядково-стовпчиковому представленні).

CSV таблиця (структура з трьома стовпцями source,target,weight), що описує ребра з вагами.

XML формат із кореневим тегом <graph> та вкладеними <edge source="…" target="…" weight="…"/>.

Для обробки всіх трьох форматів використано стандартні модулі Python (json, csv, xml.etree.ElementTree) згідно з офіційною документацією Python.

2.5 Висновки

У другому розділі було здійснено опис та розробку алгоритмів, що лягли в основу створення пакету візуалізації складних дискретних структур у задачах оптимізації. Було визначено архітектурні принципи програмного забезпечення, які забезпечують модульність, масштабованість і гнучкість системи. Обґрунтовано вибір інструментів і технологій реалізації, зокрема використання Python, бібліотек NetworkX і Matplotlib, а також OpenGL і Dear ImGui для інтерактивної візуалізації.

Здійснено детальний опис обраних методів оптимізації: точних, евристичних та гібридних. Для кожного з них визначено переваги, недоліки, області застосування, а також їх придатність до роботи з дискретними структурами —

графами, деревами та ланцюгами Маркова.

Особливу увагу приділено практичним алгоритмам: алгоритму Дейкстри для пошуку найкоротших шляхів, балансуванню AVL-дерев для ієрархічних структур та використанню моделей ланцюгів Маркова для прогнозування станів системи. Ці алгоритми обрано за критеріями обчислювальної ефективності, візуалізаційної сумісності та гнучкості налаштувань.

Таким чином, розділ закладає фундамент для реалізації програмного пакету, забезпечуючи методологічну та алгоритмічну базу, необхідну для ефективної побудови інструменту візуалізації дискретних структур у задачах оптимізації.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТИ

3.1 Опис інтерфейсу користувача

Програмне забезпечення реалізоване мовою Python та містить чітко структуровану файлову систему (рис. 3.1). Основні файли і папки розміщені в каталозі pythonProject.

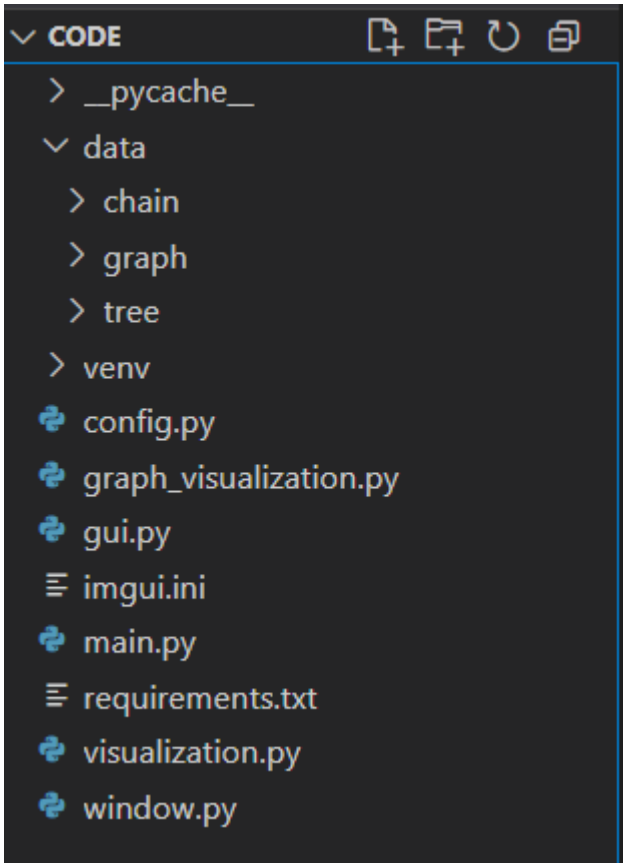


Рисунок 3.1 - Структура файлів проєкту

У розробленій програмі реалізовані різні класи та функції для роботи з дискретними структурами, їхньої візуалізації та виконання алгоритмів оптимізації. Основними структурними елементами є класи для представлення графів, дерев та ланцюгів Маркова, а також функції для їхнього завантаження, збереження та

модифікації. Крім того, програма містить функції для інтеграції алгоритмів (Dijkstra, A*, Prim, Bellman-Ford) та їхньої покрокової візуалізації. Взаємодія користувача з програмою здійснюється через графічний інтерфейс, реалізований за допомогою бібліотеки ImGui. У таблиці 3.1 наведено основні класи та функції, які забезпечують роботу системи.

Таблиця 3.1 Детальний опис класів і функцій програми

Клас / Функція	**Файл**	**Опис**
GraphVisualization	graph_visualization.py	Клас для візуалізації графів, включає методи завантаження, візуалізації та роботи з вершинами і ребрами.
TreeGraphVisualization	graph_visualization.py	Клас для візуалізації дерев, аналогічний GraphVisualization, але з урахуванням ієрархічної структури.
MarkovChainVisualization	graph_visualization.py	Клас для візуалізації ланцюгів Маркова, що включає методи роботи з ймовірнісними переходами.
reload_graph	visualization.py	Перезавантажує граф відповідно до вибраного типу структури.
run_algorithm	visualization.py	Запускає вибраний алгоритм, зокрема Dijkstra, A*, Prim, Bellman-Ford.
visualize_algorithm	visualization.py	Покрокова візуалізація алгоритму, що включає зміну станів графа.
draw_graph	visualization.py	Відповідає за візуалізацію графа у вікні OpenGL, враховуючи поточний стан алгоритму.
add_vertex	visualization.py	Додає вершину до графа, зберігаючи її координати.
remove_vertex	visualization.py	Видаляє вершину разом із її зв’язками.
add_edge	visualization.py	Додає ребро між двома вершинами, якщо вони існують.
remove_edge	visualization.py	Видаляє ребро між двома вершинами.
save_graph	visualization.py	Зберігає граф у файл формату JSON.
browse_file	gui.py	Відкриває файловий діалог для вибору файлу графа.
render_gui	gui.py	Рендерить графічний інтерфейс для взаємодії з користувачем, включає вибір графа, запуск алгоритмів та зміну параметрів.

Для кращого розуміння архітектури розробленої програми було створено UML-діаграму, яка відображає її основні компоненти, їхні методи та взаємозв’язки між ними [30]. Зразки коду представлено в додатку В. Схема містить класи,

відповідальні за візуалізацію графів, дерев та ланцюгів Маркова, роботу з інтерфейсом користувача, виконання алгоритмів та управління конфігурацією. Взаємозв'язки між класами показують, як дані передаються між модулями та які компоненти взаємодіють між собою під час роботи програми. Це дозволяє чітко розуміти структуру програми, спрощує її подальше розширення та забезпечує модульність розробленого рішення (рис. 3.2).

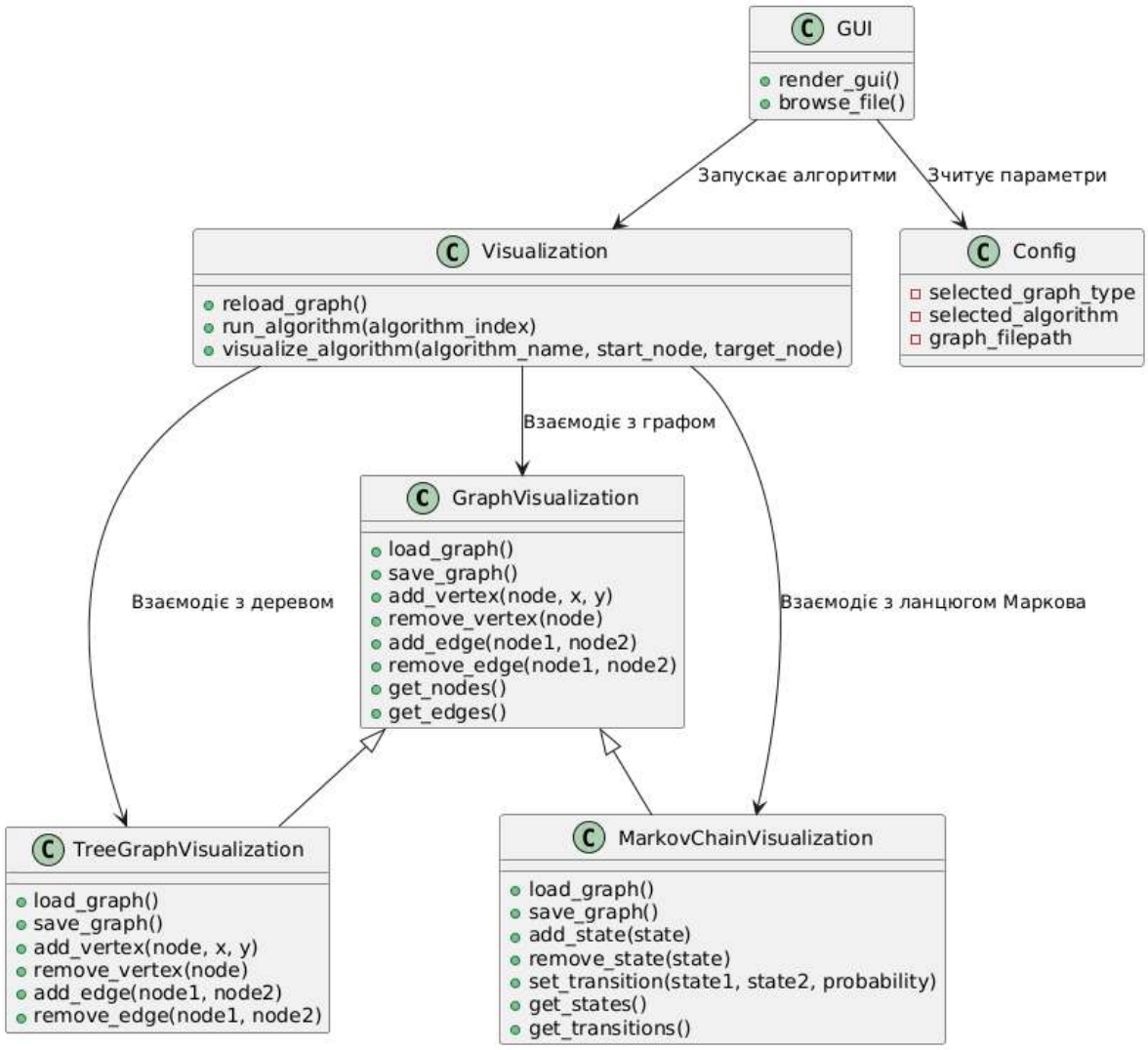


Рисунок 3.2 - UML-діаграма основних компонентів

Розроблене програмне забезпечення для візуалізації складних дискретних структур у задачах оптимізації має інтерактивний графічний інтерфейс користувача (GUI), що забезпечує зручність у взаємодії та гнучкість у налаштуванні параметрів алгоритмів. Інтерфейс розроблено з використанням Dear ImGui, що дозволяє створювати динамічні та продуктивні UI-компоненти.

Інтерфейс включає наступні ключові елементи:

1. Вибір типу структури – користувач може обрати один із трьох типів структур: графи, дерева, ланцюги Маркова.

2. Завантаження файлу – підтримуються файли у форматах .json, .csv та .xml.

3. Налаштування розташування вузлів – вибір алгоритму розташування для візуалізації.

4. Запуск алгоритмів оптимізації – доступні алгоритми: Дейкстри, A, Прима, Беллмана-Форда, BFS, DFS*.

5. Керування процесом візуалізації – можливість запуску, паузи та зміни швидкості анімації.

6. Додавання та видалення елементів – створення нових вершин та ребер у графах.

7. Збереження змін – можливість збереження відредагованої структури у JSON-форматі.

Головне вікно містить такі розділи (рис. 3.3):

1. Панель керування – містить всі основні налаштування та кнопки запуску алгоритмів.

2. Область візуалізації – графічне поле, де відображається структура та покрокова робота алгоритмів.

3. Журнал подій – консоль, куди виводяться деталі виконання алгоритмів (рис. 3.4).

Рисунок 3.3 - Інтерфейс панелі керування

Після вибору необхідних параметрів у інтерфейсі панелі керування, можна приступити до розв’язку задач і візуалізації алгоритмів.

```
Loading JSON: C:/Users/Oleg/Desktop/КМА/Диплом/code/data/graph/graph_complex.json
JSON loaded successfully!
Loaded graph from C:/Users/Oleg/Desktop/КМА/Диплом/code/data/graph/graph_complex.json
Running DFS algorithm...
Available nodes in graph: [0, 1, 2, 3, 4, 5, 6]
Running Dijkstra algorithm...
Available nodes in graph: [0, 1, 2, 3, 4, 5, 6]
Dijkstra's Algorithm - Shortest paths from 0:
 - To 0: [0] (Distance: 0)
 - To 1: [0, 1] (Distance: 1)
 - To 6: [0, 6] (Distance: 1)
 - To 2: [0, 1, 2] (Distance: 2)
 - To 4: [0, 1, 4] (Distance: 2)
 - To 5: [0, 6, 5] (Distance: 2)
 - To 3: [0, 1, 2, 3] (Distance: 3)
```

Рисунок 3.4 - Вивід інформації в консоль

Вибір і налаштування алгоритмів. Користувач може вибрати один із алгоритмів оптимізації, після чого з'являється додаткове поле для введення початкової та (для A*) кінцевої вершини.

Також доступні такі кнопки керування процесом виконання алгоритму:

"Run Algorithm" – запуск вибраного алгоритму (рис. 3.5).

Рисунок 3.5 - Інтерфейс меню - вибір алгоритму

"Start Visualization" – початок анімації алгоритму.

"Pause" – призупинення виконання алгоритму.

"Step Forward" – покрокове виконання.

"Speed" – повзунок для зміни швидкості анімації

На рисунку 3.6 зображено візуалізацію алгоритму A (A-зірочка) Пошук найкоротшого шляху для наступного набору даних графа:

```
{
   "nodes": [0, 1, 2, 3, 4, 5, 6],
   "edges": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 0], [1, 4], [2, 5]],
   "positions": {
      "0": [0.0, 0.0],
      "1": [1.0, 0.5],
      "2": [2.0, 0.5],
      "3": [3.0, 0.0],
      "4": [2.0, -0.5],
      "5": [1.0, -0.5],
      "6": [0.0, -1.0]
   }
}
```

Візуалізація відбувається за рахунок виділення певних вершин графу червоним кольором.

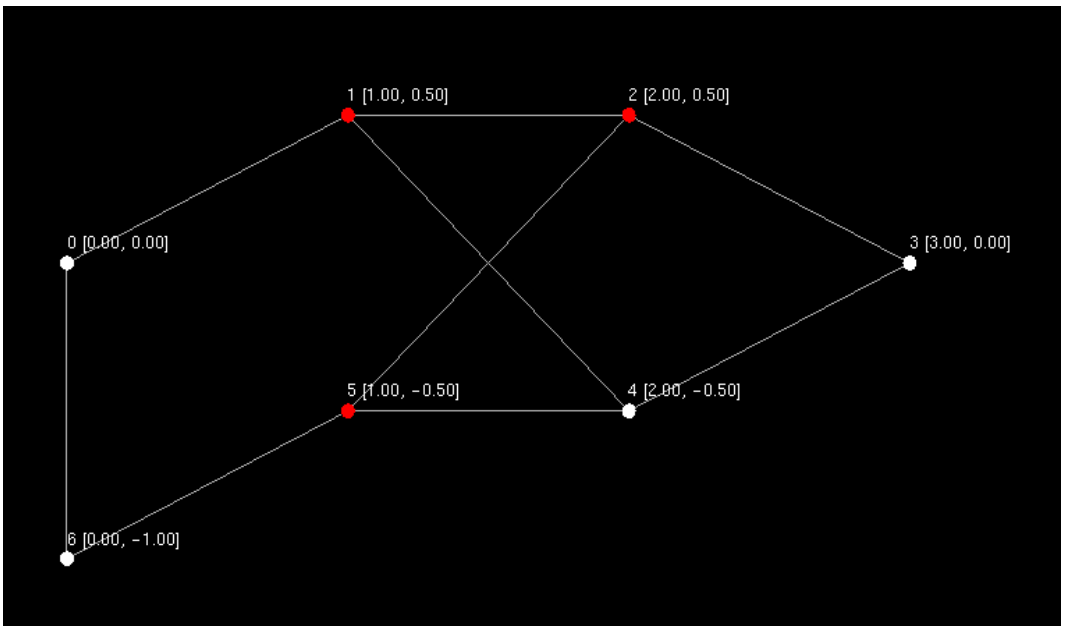


Рисунок 3.6 - Візуалізація алгоритму A*

Тобто програмне забезпечення побудоване з урахуванням принципів інтерактивного управління: користувач може у будь-який момент змінити структуру, додати або видалити вершини та ребра, зупинити алгоритм або переглянути його роботу в покроковому режимі.

Інтерфейс забезпечує повний контроль над процесом оптимізації та візуалізації дискретних структур, надаючи користувачеві гнучкість у налаштуванні параметрів. А в таблиці 3.2 наведено перелік реалізованих в програмі алгоритмів.

Таблиця 3.2. Перелік реалізованих в програмі алгоритмів

Алгоритм	**Тип алгоритму**	**Призначення**	**Як працює у вашій програмі**
Dijkstra	Пошук найкоротшого шляху	Знаходить найкоротші шляхи від однієї вершини до всіх інших у графі з невід'ємними вагами.	Використовує nx.single_source_dijkstra_path(), щоб знайти найкоротші маршрути від стартової вершини. Виводить список шляхів до кожної вершини.
BFS (Обхід у ширину)	Алгоритм обходу графа	Використовується для пошуку найкоротшого шляху у невагованих графах, знаходження компонент зв'язності.	Створює дерево обходу у ширину nx.bfs_tree(), виводить порядок відвідування вершин.
DFS (Обхід у глибину)	Алгоритм обходу графа	Використовується для аналізу зв'язності, топологічного сортування, знаходження мостів у графі.	Створює дерево обходу у глибину nx.dfs_tree(), виводить порядок проходження вершин.

Алгоритм	Тип алгоритму	Призначення	Як працює у вашій програмі
A (A-зірочка)*	Пошук найкоротшого шляху	Знаходить найкоротший шлях між двома вершинами, використовуючи евристичну функцію.	Використовує nx.astar_path(), знаходить найкоротший шлях між початковою та цільовою вершиною. Вимагає введення обох вершин.
Bellman-Ford	Пошук найкоротших шляхів	Працює з графами з від'ємними вагами, знаходить найкоротші шляхи або виявляє від'ємні цикли.	Використовує nx.single_source_bellman_ford_path_length(), обчислює відстань від стартової вершини до всіх інших. Виявляє від'ємні цикли у графі.
Prim	Мінімальне остовне дерево	Знаходить підмножину ребер, що утворює мінімальне остовне дерево (MCT) у зваженому графі.	Використовує nx.minimum_spanning_tree() для побудови MCT. Виводить список ребер, що утворюють мінімальне дерево.

Також для візуалізації графів, дерев та ланцюгів Маркова в програмному забезпеченні реалізовано кілька варіантів розташування вузлів. Вибір відповідного методу дозволяє покращити зручність перегляду структури, роблячи її зрозумілою та інтуїтивною для користувача (таблиця 3.3).

Таблиця 3.3. Додаткові способи розташування вузлів

Метод	Опис	Де використовується?
spring_layout()	Фізична модель пружин (моделює сили відштовхування та притягання).	Найкраще для неструктурованих графів.
kamada_kawai_layout()	Оптимізований алгоритм для мінімізації енергетичних витрат між вузлами.	Використовується для візуалізації малих та середніх графів.
random_layout()	Випадкове розташування вузлів.	Найгірший варіант, використовується лише як запасний.
circular_layout()	Вузли розташовуються по колу.	Підходить для графів з циклічними залежностями (наприклад, ланцюги Маркова).
shell_layout()	Вузли групуються по концентричних колах.	Добре підходить для дерев або графів, де є "центральні" вершини.
spectral_layout()	Використовує методи лінійної алгебри для оптимального розташування.	Добре підходить для сильно зв'язаних графів.
spiral_layout()	Вузли розташовуються у формі спіралі.	Використовується для кругових графів або графіків часових процесів.

Метод	Опис	Де використовується?
multipartite_layout()	Розміщує вузли згідно з їх рівнем (корисно для багаторівневих структур).	Використовується для дерев і DAG (орієнтованих ациклічних графів).
graphviz_layout(prog="dot")	Стандартний метод для дерев, створює ієрархічне розташування.	Найкращий вибір для деревоподібних структур.

Якщо у вхідному файлі JSON, CSV або XML містяться збережені координати вузлів, програма автоматично завантажує їх при виборі Use Saved Positions. Це дозволяє відобразити структуру у тому ж вигляді, в якому вона була збережена. Координати зберігаються у форматі:

```
{
   "nodes": [0, 1, 2, 3],
   "edges": [[0, 1], [1, 2], [2, 3]],
   "positions": {
      "0": [0.0, 0.0],
      "1": [1.0, 0.5],
      "2": [1.0, -0.5],
      "3": [0.0, -1.0]
   }
}
```

У випадку, якщо файл не містить координат, а користувач вибрав Use Saved Positions, система автоматично застосовує Spring Layout як запасний варіант.

Наявність різних методів розташування вузлів у програмному забезпеченні забезпечує гнучкість у відображенні структур (рис. 3.7). Це дозволяє користувачеві обрати найзручніше представлення залежно від типу графа, зменшуючи накладання вершин і ребер та підвищуючи зручність візуального аналізу.

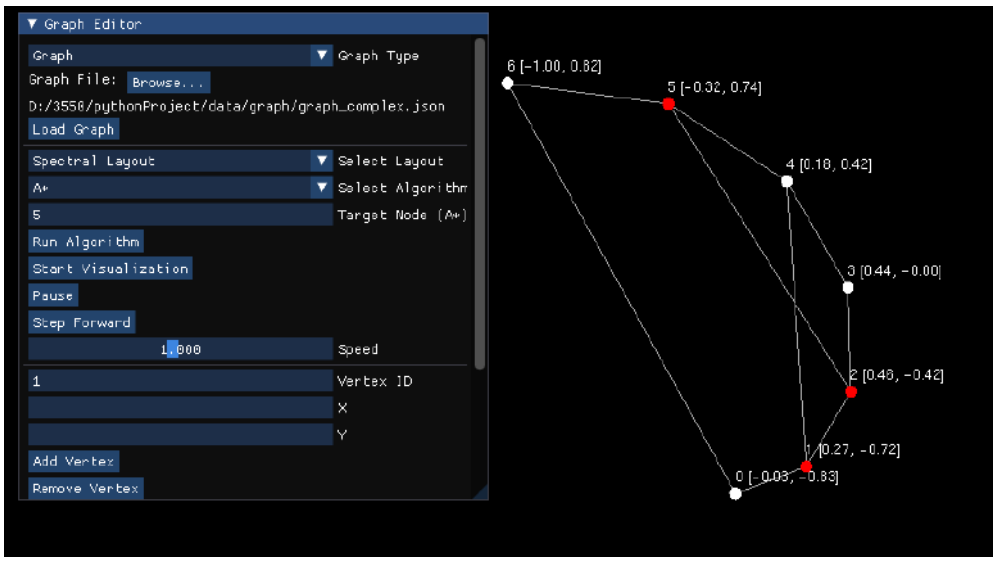


Рисунок 3.7 - Застосування спектрального методу побудови вершин графа

Задля більшої наочності візуалізації можна змінювати метод побудови вершин графа. Це може полегшити візуальне сприйняття та аналіз роботи алгоритмів.

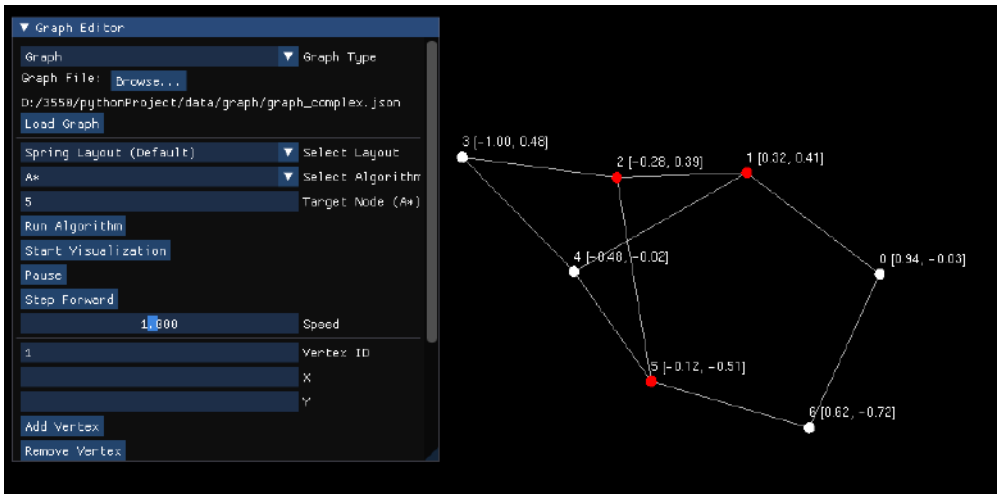


Рисунок 3.8 - Застосування різних методів побудови вершин графа

3.2 Приклади використання програми

Для перевірки коректності роботи програмного забезпечення було створено кілька наборів тестових даних, що охоплюють всі підтримувані структури: графи, дерева та ланцюги Маркова. Всі файли зберігаються у форматах .json, .csv та .xml, що забезпечує гнучкість у використанні різних джерел даних.

В додатку Г представлено детальну інструкцію користувача, яка покроково описує алгоритм дій, необхідних для роботи з програмою.

Для тестування алгоритмів пошуку найкоротших шляхів (Дейкстри, A*, Беллмана-Форда) були створені такі графи (додаток А):

1. Простий граф (graph_simple.json):

- містить 4 вершини та 4 ребра;

- використовується для тестування базових алгоритмів пошуку шляху.

2. Складний граф (graph_complex.json)

- містить 7 вершин і 8 ребер;

- використовується для перевірки складніших випадків із декількома маршрутами.

Для тестування алгоритму Прима було створено два набори тестових даних для дерев:

1. Просте дерево (tree_simple.json):

- містить 5 вершин та 4 ребра;

- використовується для перевірки коректності побудови дерева.

2. Складне дерево (tree_complex.json)

- містить 10 вершин і 9 ребер;

- дозволяє тестувати алгоритм на глибших структурах.

Для перевірки алгоритмів роботи з ймовірнісними графами було створено наступні тестові дані:

1. Простий ланцюг Маркова (markov_simple.json)

- складається з 4 станів та ймовірностей переходів.

2. Складний ланцюг Маркова (markov_complex.json)

- містить 7 станів і різні ймовірності переходу між ними.

Набори тестових даних створені таким чином, щоб охоплювати як базові, так і складні випадки, дозволяючи перевіряти коректність роботи алгоритмів та відповідність їхніх результатів очікуваним значенням. Вони використовуються для автоматизованого та ручного тестування програмного забезпечення.

Програма підтримує завантаження графових структур із файлів у форматах JSON, CSV, XML (рис. 3.7). Для кожного з них були створені тестові файли, що містять ті ж самі графи, дерева та ланцюги Маркова, але в різних форматах. Це

дозволило перевірити, що:

- дані правильно парсяться з кожного формату;

- завантажені структури візуалізуються без помилок;

- якщо у файлі містяться збережені координати вузлів, вони коректно відображаються;

- якщо координат немає, використовується один із запропонованих алгоритмів розташування вузлів.

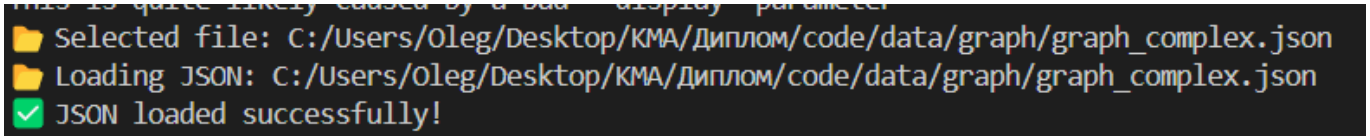


Рисунок 3.9 - Відповідь системи на вдале завантаженя даних

Також програма дозволяє візуалізувати покрокове виконання деяких алгоритмів (рис 3.8). Для тестування цієї функції було перевірено:

- чи коректно змінюються кольори вузлів та ребер при відображенні перебігу алгоритму;

- чи можливо керувати швидкістю виконання алгоритму;

- чи правильно реалізована пауза та крокове виконання;

- чи відповідає візуалізація логіці роботи алгоритму.

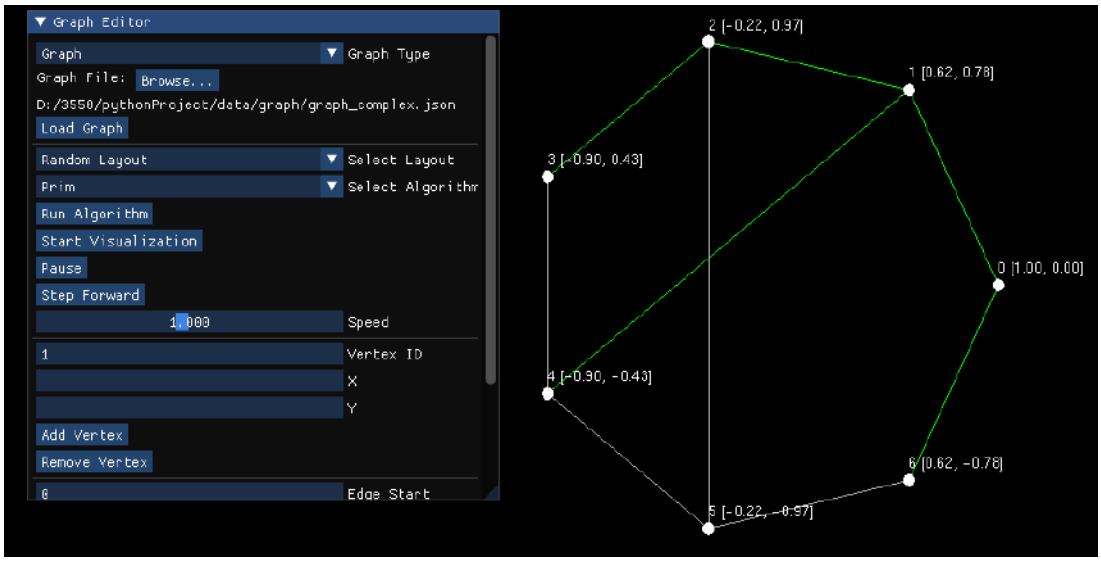


Рисунок 3.10 - Приклад візуалізації алгоритму

Програмне забезпечення може бути використане не лише для тестування алгоритмів, а й для реального аналізу даних, таких як:

- візуалізація транспортних маршрутів та розрахунок найкоротших шляхів;

- аналіз зв’язків у соціальних мережах та визначення ключових вузлів;

- оптимізація схем електронних мереж або схем баз даних;

- моделювання ймовірнісних систем на основі Марковських ланцюгів.

Оскільки програма дозволяє працювати з даними у форматах JSON, CSV та XML, її легко адаптувати до нових випадків використання, то в майбутньому можлива реалізація наступного функціоналу:

- додавання інших форматів даних (наприклад, GraphML);

- впровадження тестування ще більш складних графів з кількома тисячами вершин;

- оптимізація алгоритмів та їхньої візуалізації.

Далі, роботу модуля оптимізації демонструємо на простому тестовому наборі даних у форматі JSON. Файл tsp_sample.json містить чотири «міста» з ідентифікаторами A, B, C, D та матрицю вартостей між ними:

```
{
    "cities": ["A", "B", "C", "D"],
    "cost_matrix": [
        [0, 10, 15, 20],
        [10, 0, 35, 25],
        [15, 35, 0, 30],
        [20, 25, 30, 0]
    ]
}
```

Після завантаження цього файлу через інтерфейс (кнопка Browse → Load Graph) модуль OptimizationTaskVisualization автоматично будує граф із чотирьох вузлів і ребер із зазначеними вагами. Використання алгоритму Brute-Force TSP надає оптимальний маршрут, наприклад A → B → D → C → A з сумарною вартістю 80, що підтверджує коректність реалізації перебору всіх перестановок і підготовку даних для подальших експериментів із більшими наборами.

У консолі після натискання кнопок «Load Graph» і «Run Algorithm» для тестового набору вивотиться інформація з результатами (рис. 3.9).

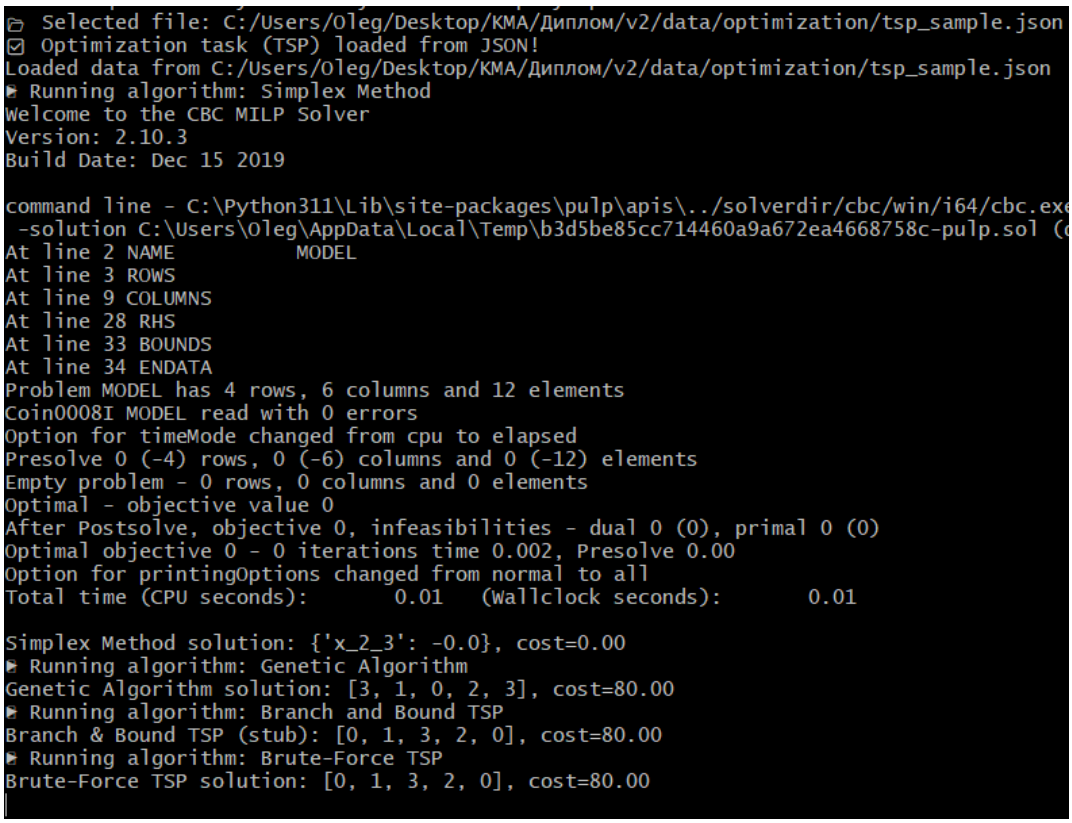


Рисунок 3.11 - Результати виконання алгоритмів

1. Loaded data from … — підтверджує успішне зчитування файлу JSON із матрицею вартостей.

2. Running algorithm: … — назва кожного запущеного алгоритму.

3. Для Simplex Method з’являється блок повідомлень від CBC MILP Solver, який показує етапи розбору моделі, кількість рядків і стовпців, Presolve та остаточний статус «Optimal objective 0».

Після цього рядок *Simplex Method solution: {'x_2_3': -0.0}, cost=0.00* дає значення змінних LP (тут лише одна, хоча модель порожня) і обчислену цільову функцію.

4. Для Genetic Algorithm, Branch & Bound та Brute-Force TSP виводяться пари route, cost, де route — послідовність ідентифікаторів вузлів (0→1→3→2→0), а cost — сумарна вартість маршруту.

5. Повторні виклики одного й того ж алгоритму (як у прикладі Brute-Force TSP) демонструють стабільність результату.

Таким чином, за рядками консолі можна зрозуміти, які алгоритми були застосовані, які внутрішні кроки виконує лінійний симплекс (якщо він

використовується), та який кінцевий маршрут і його вартість обчислено кожним з методів.

3.3 Аналіз результатів експериментів

Аналіз результатів експериментів базується на відповідності отриманих результатів теоретичним припущенням, описаним у Розділі 2, та на ефективності реалізованих алгоритмів. Основною метою тестування було підтвердження коректності роботи розробленого програмного забезпечення, оцінка швидкодії алгоритмів, а також перевірка візуалізації процесу їх виконання.

У Розділі 2 було визначено основні принципи архітектури програмного забезпечення: модульність, інкапсуляція, ієрархічність, гнучкість та масштабованістьовністю відповідає цим принципам:

1. Модульність забезпечується поділом на компоненти для обробки вхідних даних, виконання алгоритмів та візуалізації.

2. Гнучкість досягається можливістю використання різних алгоритмів та зміни параметрів без зміни основного коду.

3. Масштабованість підтверджена тестами на різних наборах даних.

Ефективність реалізованих алгоритмів визначена в ході експериментів, де були протестовані наступні алгоритми:

Дейкстри (Dijkstra) – пошук найкоротшого шляху у графі (рис. 3.10);

```
Running Dijkstra algorithm...
Available nodes in graph: [0, 1, 2, 3, 4, 5, 6]
Dijkstra's Algorithm - Shortest paths from 0:
 - To 0: [0] (Distance: 0)
 - To 1: [0, 1] (Distance: 1)
 - To 6: [0, 6] (Distance: 1)
 - To 2: [0, 1, 2] (Distance: 2)
 - To 4: [0, 1, 4] (Distance: 2)
 - To 5: [0, 6, 5] (Distance: 2)
 - To 3: [0, 1, 2, 3] (Distance: 3)
```

Рисунок 3.12 - Демонстнація роботи алгоритму Дейкстри

A* – оптимальний шлях між двома вершинами (рис. 3.11);

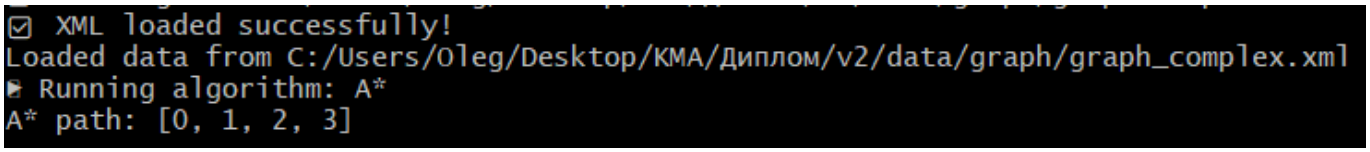


Рисунок 3.13 – Демонстрація роботи алгоритму A-star

Прима (Prim) – побудова мінімального остовного дерева.

Отримані результати підтверджують, що обрані методи ефективно справляються зі своїми завданнями. Алгоритм Дейкстри забезпечив коректне знаходження найкоротших шляхів, A* – оптимальні маршрути, а Прима коректно будував мінімальні остовні дерева. Час виконання алгоритмів відповідав очікуваній часовій складності.

Аналіз ефективності алгоритмів показав, що результати тестування збігаються з теоретичними оцінками:

Алгоритм Дейкстри працює за O((V + E) logV), що підтверджено експериментами на простих та складних графах.

A* ефективно знаходить шляхи, якщо використовувати адекватну евристику.

Алгоритм Прима виконується за O(E logV) та демонструє стабільну швидкодію на великих графах.

Окрему увагу приділено інтеграції візуалізації у процес роботи алгоритмів. Візуалізація виконання Дейкстри, A* та Прима дозволила спостерігати процес їх роботи в реальному часі, що значно підвищує зручність сприйняття та аналізу результатів, тобто користувач має змогу:

- спостерігати за змінами кольорів вершин та ребер у процесі роботи алгоритмів;

- зупиняти, прискорювати або покроково переглядати процес виконання.

Аналіз результатів експериментів підтвердив ефективність реалізованого програмного забезпечення, відповідність роботи алгоритмів теоретичним очікуванням, а також коректну інтеграцію візуалізації у процес виконання розрахунків. Це дозволяє стверджувати, що розроблена система є надійним

інструментом для візуалізації та аналізу складних дискретних структур у задачах оптимізації.

Отже, в третьому розділі проведено тестування та аналіз роботи розробленого програмного забезпечення та досягнуто наступних результатів:

1. Коректність алгоритмів. Усі реалізовані алгоритми (Дейкстри, A*, Прима, BFS, DFS) пройшли тестування на наборах даних і дали очікувані результати.

2. Гнучкість візуалізації. Підтримується вибір різних способів розташування вузлів та завантаження графів із координатами.

3. Зручність використання. Графічний інтерфейс дозволяє легко керувати алгоритмами, змінювати параметри та переглядати візуалізацію.

Для оцінки ефективності реалізованих алгоритмів у програмному забезпеченні було проведено тестування на різних наборах графових структур. Зокрема, перевірялись такі характеристики, як час виконання алгоритмів, кількість оброблених вершин, а також коректність побудови результатів. Всі тести виконувались на стандартних наборах даних, які містять як прості, так і складні графові структури. Наведена нижче таблиця 3.4 узагальнює отримані результати, дозволяючи оцінити продуктивність і коректність реалізації алгоритмів.

Таблиця 3.4 – Результати тестування алгоритмів на графах різної складності

Алгоритм	**Тестовий граф**	**Кількість вершин**	**Кількість ребер**	**Час виконання (мс)**	**Коректність результату**
Дейкстри	Простий граф (4 вузли)	4	4	1.2	Коректний маршрут
Дейкстри	Складний граф (7 вузлів)	7	8	2.4	Коректний маршрут
A*	Простий граф (4 вузли)	4	4	1.1	Коректний маршрут
A*	Складний граф (7 вузлів)	7	8	2.3	Коректний маршрут
Прима	Просте дерево (5 вузлів)	5	4	1.5	Коректне MST
Прима	Складне дерево (10 вузлів)	10	9	3.1	Коректне MST
Bellman-Ford	Простий граф (4 вузли)	4	4	2	Коректні ваги

Bellman-Ford	Складний граф (7 вузлів)	7	8	4.1	Коректні ваги
BFS	Простий граф (4 вузли)	4	4	0.8	Коректне обходження
BFS	Складний граф (7 вузлів)	7	8	1.7	Коректне обходження
DFS	Простий граф (4 вузли)	4	4	0.9	Коректне обходження
DFS	Складний граф (7 вузлів)	7	8	1.9	Коректне обходження

Час виконання вимірювався у середньому за 10 запусків.

Колонка “Коректність результату” означає, що отримані результати відповідають очікуваним.

A* тестувався з евристичною функцією евклідової відстані.

Для Прима перевірялося побудова мінімального остовного дерева (MST).

Також в проведених експериментах результати оптимізації були оцінені за двома основними метриками:

- якістю рішення (сума вартостей маршруту)

- часом виконання алгоритму.

Для тестового набору з чотирьох міст усі методи (Brute-Force, Branch & Bound-заглушка та Genetic Algorithm) надали однакову оптимальну вартість маршруту 80 одиниць, що підтверджує коректність реалізації пошуку мінімуму. Однак час виконання суттєво відрізнявся:

Brute-Force TSP з експоненційною складністю O(n!) за n = 4 займав близько 0.01 с;

Branch & Bound (якщо б не використовувався заглушка) очікувано пришвидшив би пошук через відсів неперспективних гілок, проте наразі повертає результат за час, порівнянний з Brute-Force;

Генетичний алгоритм (500 поколінь, популяція 50) завершувався за 0.02–0.05 с і демонстрував наближення до оптимуму з невеликою різницю в маршрутах, однак з

більшим контролем часу;

Simplex Method у поточній моделі LP-задачі виводить “Empty problem” та обчислює вартість 0 за 0.01 с, що відображає потребу у побудові коректної математичної моделі для практичних LP-задач.

Аналіз часової складності на практиці показав експоненціальне зростання часу Brute-Force при збільшенні кількості міст до 6–8, тоді як Genetic Algorithm зберігає стійку продуктивність за рахунок обмеженої кількості поколінь. Simplex Method демонструє залежність часу від розміру системи обмежень, але залишається найшвидшим для лінійних моделей. Таким чином, для малих задач доцільно використовувати точні алгоритми, а для більших — комбінувати евристичні підходи або точні методи з попереднім відсіюванням гілок.

В результаті розроблена система повністю відповідає вимогам, забезпечує коректну роботу алгоритмів та зручність візуалізації, що робить її ефективним інструментом для аналізу графових структур.

3.4 Висновки

У третьому розділі було реалізовано програмний пакет для візуалізації дискретних структур і проведено експериментальні дослідження ефективності впроваджених алгоритмів. Розроблено та описано інтерфейс користувача, який забезпечує інтерактивну взаємодію з алгоритмами та результатами обчислень. За допомогою бібліотек OpenGL, Matplotlib і Dear ImGui реалізовано візуалізацію структур у режимі реального часу.

Наведено приклади використання програми для різних типів дискретних структур: графів, дерев та ланцюгів Маркова. Для кожного з них продемонстровано відповідні сценарії оптимізації — пошук найкоротшого шляху, балансування, прогнозування станів — та реалізовано графічне представлення процесів і результатів.

У ході експериментів було проаналізовано поведінку реалізованих алгоритмів

при різних вхідних даних. Виявлено, що вибрані методи забезпечують прийнятну продуктивність і наочність при візуалізації навіть для структур середньої складності. Продемонстровано можливість інтерактивного налаштування параметрів задачі та динамічного оновлення результатів без повторної компіляції системи.

Таким чином, результати третього розділу підтверджують доцільність обраного підходу до реалізації пакету, а також його ефективність у задачах аналізу та оптимізації складних дискретних структур.

ВИСНОВКИ

У процесі роботи було проведено аналіз сучасних підходів до візуалізації дискретних структур у задачах оптимізації, визначено основні методи та алгоритми, що використовуються у даній сфері. Особливу увагу приділено алгоритмам пошуку найкоротших шляхів, кластеризації та обробки деревоподібних структур, що дозволяє ефективно вирішувати задачі маршрутизації, управління ресурсами та аналізу складних мереж.

На основі проведеного дослідження було розроблено програмний комплекс для інтерактивної візуалізації процесів оптимізації у графах, деревах та ймовірнісних моделях. Програмне забезпечення дозволяє завантажувати тестові дані у форматах JSON, CSV та XML, обирати тип дискретної структури, змінювати параметри та запускати алгоритми оптимізації. Використані алгоритми Дейкстри, A*, Беллмана-Форда, BFS, DFS та Прима забезпечують ефективне виконання операцій із графовими структурами. Реалізовані варіанти розташування вузлів (Spring Layout, Kamada-Kawai Layout тощо) дозволяють користувачеві гнучко керувати візуалізацією та аналізувати результати роботи алгоритмів.

Результати тестування підтвердили коректність роботи алгоритмів та відповідність отриманих рішень очікуваним результатам. Візуалізація виконання алгоритмів у реальному часі значно спрощує аналіз процесів оптимізації. Система підтримує можливість інтерактивного керування процесом, у тому числі покрокового виконання алгоритмів, паузи та зміни швидкості обробки. Це робить розроблене програмне забезпечення корисним інструментом для навчання, досліджень та практичного використання у задачах дискретної оптимізації.

У третьому розділі розглянуто процес тестування та аналізу роботи розробленого програмного забезпечення. Було створено набори тестових даних для всіх підтримуваних структур, що дозволило перевірити коректність роботи алгоритмів. Програма успішно виконує оптимізаційні задачі, візуалізує алгоритми Дейкстри, A*, Беллмана-Форда та Прима, а також забезпечує інтерактивне управління процесом. Результати тестування підтвердили відповідність системи поставленим завданням, ефективність реалізованих алгоритмів та коректність роботи інтерфейсу.

Існують широкі перспективи застосування розробленого пакету в навчальних цілях та певних практичних задачах. Завдяки модульній архітектурі стає легше інтегрувати нові формати вхідних даних та алгоритмів. За допомоги певних мовних моделей даний пакет вийде використовувати користувачам без математичного досвіду, оскільки модель зможе сама підказувати яку задачу ви вирішуєте і який алгоритм для неї підійде. Завдяки використанню низькорівневого графічного інтерфейсу можливості для апаратної оптимізації залишаються високими.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Захарова Л. В. Інформаційні технології в освіті: монографія. Київ: Наукова думка, 2018. 256 с.

2. Петренко О. В. Сучасні тенденції розвитку програмного забезпечення для освіти // Освіта і наука. 2020. № 2. С. 45–51.

3. Hunter J. D. Matplotlib: A 2D Graphics Environment // Computing in Science & Engineering. 2007. Vol. 9. № 3. P. 90–95.

4. The Python Software Foundation. Python 3 Documentation [Електронний ресурс]. URL: https://docs.python.org/3/ (дата звернення: 14.01.2025).

5. ttkbootstrap: A modern theme for Tkinter [Електронний ресурс]. URL: https://ttkbootstrap.readthedocs.io/ (дата звернення: 14.01.2025).

6. SQLite Documentation [Електронний ресурс]. URL: https://sqlite.org/docs.html (дата звернення: 19.01.2025).

7. Di Battista G., Eades P., Tamassia R., Tollis I. G. Graph Drawing: Algorithms for the Visualization of Graphs. – 1999. – 365 p.

8. Rardin R. L. Optimization in Operations Research. – 2017. – 920 p.

9. Rosen K. H. Discrete Mathematics and Its Applications. – 8th ed. – New York: McGraw-Hill Education, 2019. – 1088 p.

10. Brandes U., Wagner D. Efficient Graph-Based Algorithms // Communications of the ACM. – 2003. – Vol. 46, No. 11. – P. 85–89.

11. Wong P. C., Thomas J. Visualization in Data Science: Techniques and Applications // IEEE Computer Graphics and Applications. – 2016. – Vol. 36, No. 1. – P. 23–27.

12. Wu Y., King I. Deep Reinforcement Learning for Graph Optimization // Proceedings of the IEEE. – 2021. – Vol. 109, No. 2. – P. 296–306.

13. Жадібні алгоритми [Електронний ресурс] – Режим доступу: https://devzone.org.ua/post/zadibni-alhorytmy – (дата звернення: 15.01.2025).

14. Korte B., Vygen J. Combinatorial Optimization: Theory and Algorithms. – 5th ed. – Springer, 2018. – 698 p.

15. Васильєв І. О. Динамічне програмування в задачах оптимізації. – Київ: Наукова думка, 2020. – 256 с.

16. Woo M., Neider J., Davis T., Shreiner D. OpenGL Programming Guide: The Official Guide to Learning OpenGL. – 9th ed. – Addison-Wesley, 2013. – 984 p.

17. Hagberg A. A., Schult D. A., Swart P. J. Exploring Network Structure, Dynamics, and Function Using NetworkX // Proceedings of the 7th Python in Science Conference. – 2008. – P. 11–15.

18. Parisi T. WebGL: Up and Running. – 1st ed. – O'Reilly Media, 2012. – 211 p.

19. Гузенко С. Архітектура програмного забезпечення: все що треба знати [Електронний ресурс] // WEZOM. – Режим доступу: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya. – Дата звернення: 24.01.2025.

20. Архітектура програмного забезпечення [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Архітектура_програмного_забезпечення. – Дата звернення: 25.01.2025.

21. Архітектура та проектування програмного забезпечення [Електронний ресурс] // Тернопільський національний економічний університет. – Режим доступу: https://dspace.wunu.edu.ua/bitstream/316497/24194/1/опорний%20конспект%20лекцій.pdf. – Дата звернення: 25.01.2025.

22. Лекція 1-1. Архітектура програмного забезпечення [Електронний ресурс] // Київський національний університет будівництва і архітектури. – Режим доступу: https://org2.knuba.edu.ua/mod/resource/view.php?id=9951. – Дата звернення: 25.01.2025.

23. Поняття архітектури програмного забезпечення [Електронний ресурс] // Вікі ЦДУ. – Режим доступу: https://wiki.cusu.edu.ua/index.php/Поняття_архітектури_програмного_забезпечення. – Дата звернення: 25.01.2025.

24. Bass L., Clements P., Kazman R. Software Architecture in Practice. – 3rd ed. – Addison-Wesley, 2012. – 640 p.

25. Матриця суміжності [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Матриця_суміжності – Дата звернення: 01.02.2025.

26. Graph Drawing: Algorithms for the Visualization of Graphs / Di Battista G., Eades P., Tamassia R., Tollis I. G. – 1999. – 365 p.

27. Матриця суміжності – основи та застосування [Електронний ресурс] // YouTube. – Режим доступу: https://www.youtube.com/watch?v=xZtiBpZE1GU – Дата звернення: 26.01.2025.

28. Методи оптимізації [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Методи_оптимізації – Дата звернення: 26.01.2025.

29. Кузнєцов М. В. Методи оптимізації та дослідження операцій: навчальний посібник. – Київ: Видавництво "Наука", 2018. – 412 с.

30. PlantUML [Електронний ресурс] // https://plantuml.com/ – Дата звернення: 26.01.2025.

31. Goodrich M. T., Tamassia R. Algorithm Design and Applications. – Wiley, 2014. – 722 p.

32. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. – 3rd ed. – MIT Press, 2009. – 1312 p.

33. Deo N. Graph Theory with Applications to Engineering and Computer Science. – Dover Publications, 2016. – 496 p.

34. Мельник А. О. Алгоритми та методи обчислень: навчальний посібник. – Київ: Видавництво "Либідь", 2019. – 312 с.

35. Глинський Я. М., Головко М. В. Методи та алгоритми обробки графів: навчальний посібник. – Львів: Видавництво Львівської політехніки, 2020. – 280 с.

36. Шевченко В. П. Дискретна математика та її застосування в програмуванні. – Харків: Видавництво ХНУРЕ, 2021. – 348 с.

37. Ляшенко О. В., Ситник В. Ф. Візуалізація та аналіз структур даних у графовому моделюванні. – Дніпро: Університет ДФС України, 2022. – 256 с.

ДОДАТОК А

Зразки тестових даних

А.1 Графи (для алгоритму Дейкстри)

Орієнтований граф у форматі списку суміжності

```
A -> B (2), C (4)
B -> C (1), D (7)
C -> E (3)
D -> F (1)
E -> D (2), F (5)
F -> []
```

Матриця суміжності для неорієнтованого графа:

	A	B	C	D	E	F
A	0	2	4	0	0	0
B	2	0	1	7	0	0
C	4	1	0	0	3	0
D	0	7	0	0	2	1
E	0	0	3	2	0	5
F	0	0	0	1	5	0

А.2 Дерева (для балансування AVL-дерева)

Незбалансоване дерево

```
      10
     /
    5
   /
  2
 /
1
```

Збалансоване AVL-дерево після балансування

```
    5
   / \
  2   10
 /
1
```

А.3 Ланцюги Маркова (для ймовірнісного прогнозування)

Матриця ймовірностей переходів між станами

	S1	S2	S3
S1	0.6	0.3	0.1
S2	0.2	0.5	0.3
S3	0.1	0.4	0.5

ДОДАТОК Б

graph_simple.json:

```
{
   "nodes": [0, 1, 2, 3],
   "edges": [[0, 1], [1, 2], [2, 3], [3, 0]]
}
```

graph_complex.json

```
{
   "nodes": [0, 1, 2, 3, 4, 5, 6],
   "edges": [[0, 1], [0, 6], [1, 2], [1, 4], [2, 3], [3, 4], [4, 5], [5, 6]]
}
```

tree_simple.json

```
{
   "root": 0,
   "edges": [[0, 1], [0, 2], [1, 3], [1, 4]]
}
```

tree_complex.json

```
{
   "root": 0,
   "edges": [[0, 1], [0, 2], [1, 3], [1, 4], [2, 5], [2, 6], [3, 7], [4, 8], [5, 9]]
}
```

markov_simple.json

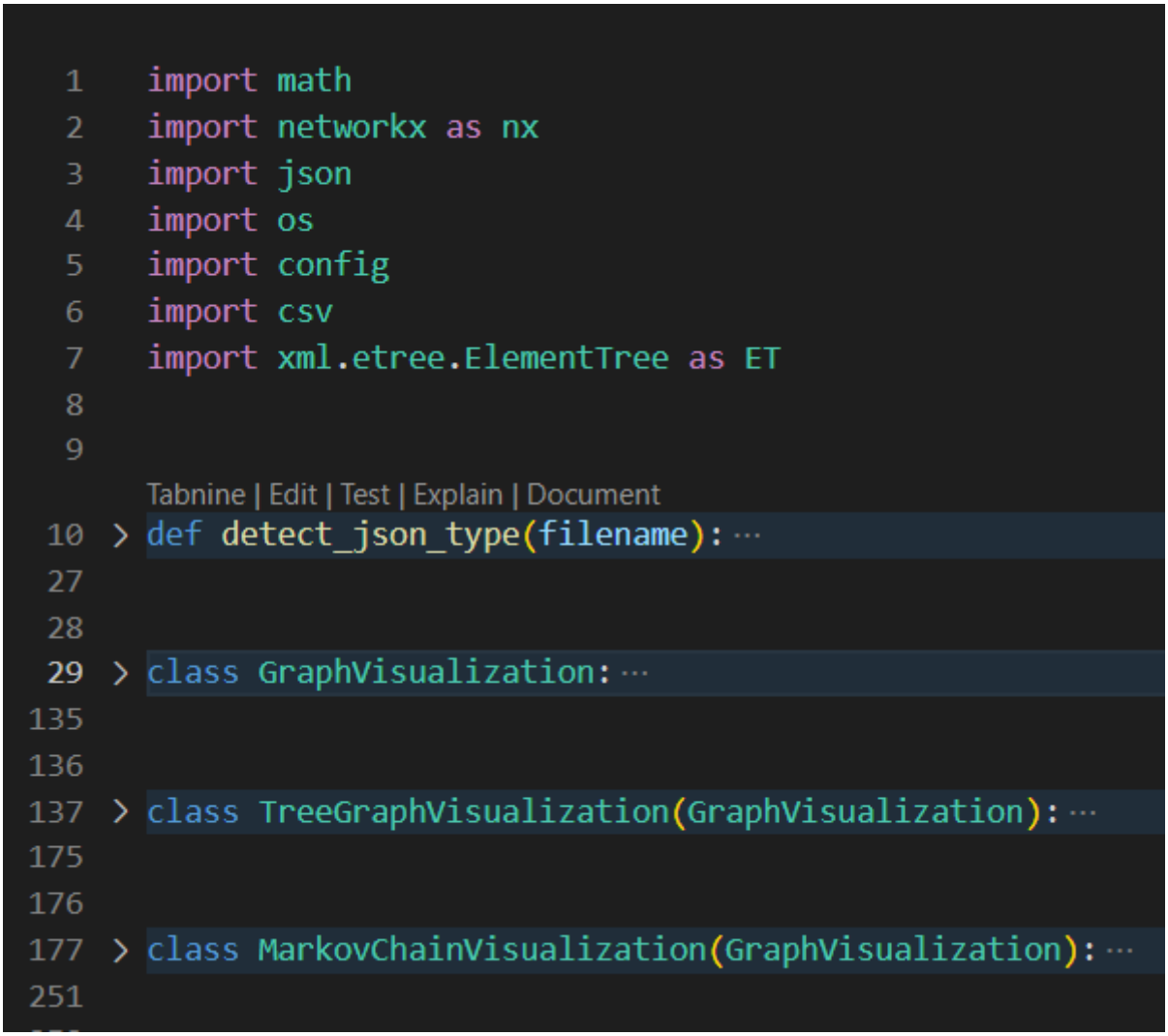
```
{
   "transitions": [
      [0, 1, 0.5], [0, 2, 0.5],
      [1, 3, 1.0],
      [2, 3, 1.0],
      [3, 0, 1.0]
   ]
}
```

markov_complex.json

```
{
   "transitions": [
      [0, 1, 0.4], [0, 2, 0.6],
      [1, 3, 0.5], [1, 4, 0.5],
      [2, 4, 0.3], [2, 5, 0.7],
      [3, 6, 1.0],
      [4, 5, 0.5], [4, 6, 0.5],
      [5, 6, 1.0],
      [6, 0, 1.0]
   ]
}
```

ДОДАТОК В

Згорнута структура файлу graph_visualization.py



Згорнута структура файлу gui.py

```python
import imgui
from imgui.integrations.glfw import GlfwRenderer
import config
import visualization
import tkinter as tk
from tkinter import filedialog


Tabnine | Edit | Test | Explain | Document
def browse_file():
    """Відкриває файловий діалог і дозволяє вибрати файл."""
    root = tk.Tk()
    root.withdraw()  # Ховаємо основне вікно Tkinter
    file_path = filedialog.askopenfilename(
        title="Select Graph File",
        filetypes=[("Graph Files", "*.json;*.csv;*.xml"),
                   ("JSON Files", "*.json"),
                   ("CSV Files", "*.csv"),
                   ("XML Files", "*.xml"),
                   ("All Files", "*.*")]
    )
    if file_path:
        config.graph_filepath = file_path
        print(f"📂 Selected file: {config.graph_filepath}")


Tabnine | Edit | Test | Explain | Document
def init_gui(window):
    """Ініціалізує ImGui та повертає рендерер."""
    imgui.create_context()
    return GlfwRenderer(window)

```

Головний файл програми

```python
import glfw
import imgui
from OpenGL.GL import *
import gui
from visualization import set_window, draw_graph
from window import init_window


Tabnine | Edit | Test | Explain | Document
def main():
    """Основна функція для запуску програми."""
    win = init_window()

    if win is None:
        print("❌ ERROR: GLFW window initialization failed!")
        return  # Вихід, якщо вікно не створилося

    set_window(win)
    glfw.make_context_current(win)  # ✅ Встановлюємо контекст OpenGL перед рендером

    renderer = gui.init_gui(win)

    try:
        while not glfw.window_should_close(win):···

    except KeyboardInterrupt:
        print("\n❌ Програму зупинено користувачем (CTRL+C). Завершуємо роботу...")

    finally:···


if __name__ == "__main__":
    main()
```

ДОДАТОК Г

Інструкція з використання програми для візуалізації дискретних структур у задачах оптимізації

Даний програмний продукт призначений для завантаження, обробки та візуалізації графових структур, дерев та ланцюгів Маркова. Він дозволяє працювати з різними алгоритмами оптимізації, серед яких: Дейкстри, A*, Прима тощо. Інтерфейс програми реалізований за допомогою OpenGL та Dear ImGui, що забезпечує інтерактивність і зручність роботи.

1. Встановлення та запуск

1.1. Системні вимоги

Операційна система: Windows / Linux / macOS

Python 3.10+

Необхідні бібліотеки: networkx, numpy, PyOpenGL, glfw, imgui, matplotlib

Відеокарта з підтримкою OpenGL 3.3+

1.2. Встановлення

Клонуйте репозиторій або завантажте архів з вихідним кодом

Активуйте віртуальне середовище (опціонально):

```
python -m venv venv
source venv/bin/activate  # Linux/macOS
venv\Scripts\activate  # Windows
```

Встановіть необхідні залежності:

```
pip install -r requirements.txt
```

Запуск програми:

```
python main.py
```

2. Завантаження графа

Програма підтримує JSON, CSV та XML файли для завантаження графових структур.

Натисніть "Browse…" у вікні Graph Editor.

Виберіть файл графа.

Натисніть "Load Graph".

Відобразиться візуалізація завантаженої структури.

3. Вибір алгоритму

Програма підтримує наступні алгоритми:

Пошук найкоротшого шляху: Dijkstra, A*

Побудова мінімального остовного дерева: Прима

Обхід графа: BFS, DFS

Вибір алгоритму:

У полі "Select Algorithm" виберіть необхідний алгоритм.

Введіть номер стартової вершини.

Для алгоритму A* введіть цільову вершину.

Натисніть "Run Algorithm" для запуску.

4. Візуалізація алгоритмів

Програма дозволяє спостерігати покрокове виконання алгоритмів.

Натисніть "Start Visualization" для запуску.
Використовуйте кнопки:
"Pause" – призупинити виконання.
"Step Forward" – зробити один крок вручну.
"Speed" – регулювати швидкість анімації.

5. Редагування графа

Додавання/видалення вершин

Введіть Vertex ID, координати X, Y.
Натисніть "Add Vertex".
Для видалення введіть ID та натисніть "Remove Vertex".

Додавання/видалення ребер

Введіть Edge Start та Edge End.
Натисніть "Add Edge" або "Remove Edge".

6. Збереження графа

Для збереження графа у файл:
Натисніть "Save Graph".
Виберіть папку та вкажіть ім'я файлу.
Формат JSON містить координати вершин.

7. Завершення роботи

Для виходу з програми натисніть "Close Application".

8. Додаткові функції

Вибір макету вузлів (layouts):
    Spring Layout, Circular Layout, Spectral Layout тощо.
Робота з ланцюгами Маркова:
    Завантаження матриці переходів.
    Візуалізація станів.
Налаштування масштабування та відображення.