НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

“До захисту допущено”
Завідувач кафедри
_____ Наталія АУШЕВА
“___” _____2026 р.

Дипломна робота
на здобуття ступеня бакалавра

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему:“Вебсистема підтримки прийняття рішень на основі методів оптимізації”

Виконала: студентка 4 курсу, групи ТР-23

Скакодуб Світлана Олександрівна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник:

професор каф. ЦТЕ, д.т.н., проф. Сліпченко Володимир Георгійович _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент:

доц. каф. ТАЕ, к.т.н., доц. Пешко Віталій Анатолійович _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Н.контроль асистент каф. ЦТЕ, д.ф. Мельниченко Артем Васильович _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ — 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ
Рівень вищої освіти — перший (бакалаврський)
Спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
“__”_____2026 р.

З А В Д А Н Н Я
на дипломну роботу студенту

Скакодуб Світлані Олександрівні
(прізвище, ім’я, по батькові)

1. Тема роботи “Вебсистема підтримки прийняття рішень на основі методів оптимізації”

керівник роботи Сліпченко Володимир Георгійович, д.т.н., проф.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “28” травня 2026 р. № 1992-с

2. Термін подання роботи студентом 04.06.2026 р.

3. Вихідні дані до роботи мови програмування Python та TypeScript; серверний фреймворк FastAPI; клієнтський фреймворк Next.js на бібліотеці React; бібліотека чисельних обчислень NumPy; реляційна система керування базами даних PostgreSQL; бібліотека компонентів інтерфейсу shadcn/ui; система контролю версій Git; середовище розробки Visual Studio Code.

4. Перелік питань, які потрібно розробити 1) проаналізувати прикладні задачі оптимізації та наявні програмні засоби, визначити неперекриті ніші; 2) розглянути математичні постановки й методи розв’язання п’яти класів задач оптимізації; 3) спроєктувати архітектуру вебсистеми: серверну, клієнтську частини, сховище даних і REST API; 4) реалізувати обчислювальне ядро з фіксацією проміжних ітерацій для покрокової візуалізації; 5) реалізувати сервіс автоматичної класифікації типу задачі та рекомендації методу розв’язання; 6) розробити клієнтський інтерфейс із покроковою візуалізацією, графічною побудовою

допустимої області та аналізом чутливості; 7) реалізувати автентифікацію користувачів, збереження задач і генерацію PDF-звітів; 8) виконати тестування та крос-верифікацію реалізованих методів

5. Орієнтовний перелік ілюстративного матеріалу компонентна діаграма системи; логічна модель бази даних; форми введення задач різних класів; сторінка результату розв'язання задачі; покрокова візуалізація симплекс-методу та методу гілок і меж; графічна побудова допустимої області задачі лінійного програмування; панель аналізу чутливості; приклад згенерованого PDF-звіту.

6. Дата видачі завдання 28.05.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Затвердження теми дипломної роботи	07.02.2026	Виконано
2	Аналіз прикладних задач оптимізації та огляд наявних засобів	28.02.2026	Виконано
3	Дослідження методів оптимізації та вибір технологій	20.03.2026	Виконано
4	Проєктування архітектури системи та бази даних	05.04.2026	Виконано
5	Програмна реалізація обчислювального ядра та REST API	25.04.2026	Виконано
6	Розробка клієнтського інтерфейсу з покроковою візуалізацією	08.05.2026	Виконано
7	Тестування та крос-верифікація реалізованих методів	13.05.2026	Виконано
8	Оформлення пояснювальної записки	15.05.26	Виконано
9	Передзахист	03.06.26	Виконано
10	Захист дипломної роботи	15 – 19.06.2026	Виконано

Студент _____ Світлана СКАКОДУБ
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Володимир СЛІПЧЕНКО
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 72 сторінках, містить 18 ілюстрацій, 3 таблиці, 2 додатки та 22 джерела в переліку посилань.

Метою роботи є створення веборієнтованої системи підтримки прийняття рішень для формулювання й автоматизованого розв’язання типових задач математичного програмування п’яти класів у єдиному браузерному інтерфейсі з покроковою візуалізацією обчислень.

Основний зміст роботи включає аналіз прикладних задач оптимізації та наявних засобів, розгляд методів розв’язання п’яти класів задач, проєктування архітектури вебсистеми та її програмну реалізацію, а також опис роботи користувача із системою.

Методи та засоби: класичні методи дослідження операцій (симплекс-метод, метод гілок і меж, метод потенціалів, угорський алгоритм, динамічне програмування), реалізовані самостійно засобами NumPy без зовнішніх солверів; серверна частина — Python і FastAPI, клієнтська — Next.js, React і TypeScript, сховище даних — PostgreSQL.

Результат: програмний застосунок, що приймає задачу одного з п’яти класів, автоматично визначає її тип, рекомендує метод, виконує обчислення з фіксацією проміжних ітерацій для покрокової візуалізації та зберігає процес у персональній історії з аналізом чутливості й генерацією PDF-звітів.

Ключові слова: СИСТЕМА ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ, МЕТОДИ ОПТИМІЗАЦІЇ, ЛІНІЙНЕ ПРОГРАМУВАННЯ, СИМПЛЕКС-МЕТОД, ТРАНСПОРТНА ЗАДАЧА, ВЕБЗАСТОСУНОК, ПОКРОКОВА ВІЗУАЛІЗАЦІЯ.

ABSTRACT

The diploma project consists of 72 pages and includes 18 illustrations, 3 tables, 2 appendices, and 22 references.

The aim of the study is to develop a web-based decision support system for formulating and automatically solving typical mathematical programming problems of five classes within a single browser interface with step-by-step visualization of computations.

The main content includes an analysis of applied optimization problems and existing tools, a review of solution methods for five problem classes, the design of the web system architecture and its software implementation, and a description of user interaction with the system.

Methods and tools: classical operations research methods (the simplex method, branch-and-bound, the method of potentials, the Hungarian algorithm, dynamic programming) implemented from scratch with NumPy without external solvers; the back end uses Python and FastAPI, the front end Next.js, React, and TypeScript, and data storage uses PostgreSQL.

Result: a software application that accepts a problem of one of the five classes, automatically determines its type, recommends a method, performs the computation while recording intermediate iterations for step-by-step visualization, and stores the process in a personal history with sensitivity analysis and PDF report generation.

Keywords: DECISION SUPPORT SYSTEM, OPTIMIZATION METHODS, LINEAR PROGRAMMING, SIMPLEX METHOD, TRANSPORTATION PROBLEM, WEB APPLICATION, STEP-BY-STEP VISUALIZATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЛП — лінійне програмування

СКБД — система керування базами даних

ЦЛП — цілочислове лінійне програмування

API — application programming interface, прикладний програмний інтерфейс

HTTP — hypertext transfer protocol, протокол передавання гіпертексту

JSON — JavaScript object notation, текстовий формат подання структурованих даних

JSONB — binary JSON, двійкове подання JSON у PostgreSQL з підтримкою індексування

JWT — JSON Web Token, компактний маркер автентифікації у форматі JSON

LP — linear programming, лінійне програмування MILP — mixed-integer linear programming, змішане цілочислове лінійне програмування

MODI — modified distribution method, метод модифікованих розподілів (метод потенціалів)

ORM — object-relational mapping, об’єктно-реляційне відображення

PDF — portable document format, формат документів зі сталим відображенням

REST — representational state transfer, архітектурний стиль проєктування вебсервісів

UUID — universally unique identifier, універсальний унікальний ідентифікатор

ВСТУП

Актуальність теми. Задачі оптимального розподілу обмежених ресурсів — планування виробництва, перевезень, призначення виконавців, відбору варіантів — постають як регулярна управлінська рутина в плановій, виробничій і логістичній діяльності. Класичні методи їх розв’язання (симплекс-метод, метод гілок і меж, метод потенціалів, угорський алгоритм, динамічне програмування) добре відомі, проте наявні програмні засоби не утворюють інструмента, однаково придатного і практику, і студентові.

Промислові алгебраїчні мови та солвери (AMPL, GAMS, Gurobi, CPLEX) вимагають фахової підготовки з дослідження операцій; відкриті бібліотеки (Pyomo, PuLP) — програмістських навичок; вбудований “Пошук рішення” Excel і веб-калькулятори окремих методів не забезпечують прозорості процесу обчислень, ведення історії розв’язань та єдиного інтерфейсу для задач різних класів. Неперекритою лишається ніша інтегрованої, доступної без встановлення ПЗ і прозорої щодо проміжних кроків системи — що й зумовлює актуальність роботи.

Мета роботи — створення веборієнтованої системи підтримки прийняття рішень для формулювання й автоматизованого розв’язання типових задач математичного програмування п’яти класів у єдиному браузерному інтерфейсі з покроковою візуалізацією процесу обчислень.

Для досягнення поставленої мети сформульовано такі завдання:

1) проаналізувати прикладні задачі оптимізації та наявні програмні засоби, визначити неперекриті ніші;

2) розглянути математичні постановки й методи розв’язання п’яти класів задач — лінійного та цілочислового програмування, транспортної задачі, задачі про призначення й задачі про рюкзак;

3) спроєктувати архітектуру вебсистеми: серверну частину, клієнтську частину, сховище даних і REST API;

4) реалізувати обчислювальне ядро з фіксацією проміжних ітерацій для

покрокової візуалізації;

5) реалізувати сервіс автоматичної класифікації типу задачі й рекомендації методу розв’язання;

6) розробити клієнтський інтерфейс із покроковою візуалізацією, графічною побудовою допустимої області, аналізом чутливості та персональною історією розв’язань;

7) реалізувати автентифікацію користувачів, збереження задач і генерацію PDF-звітів;

8) виконати тестування та крос-верифікацію реалізованих методів.

Методи та засоби. Класичні методи дослідження операцій (симплекс-метод з технікою Big-M і правилом Бленда, метод гілок і меж, метод потенціалів, угорський алгоритм, динамічне програмування), реалізовані самостійно на бібліотеці NumPy без зовнішніх готових солверів; серверна частина — мовою Python з фреймворком FastAPI; клієнтська — на Next.js, React і TypeScript; сховище даних — реляційна СКБД PostgreSQL.

Практичне значення**.** Систему орієнтовано на дві категорії користувачів: фахівців-практиків планових, логістичних та аналітичних підрозділів підприємств і студентів та викладачів технічних спеціальностей, для яких вона слугує навчальною платформою для порівняння поведінки класичних алгоритмів оптимізації на спільних вихідних даних.

Апробація результатів роботи. Основні положення та результати роботи оприлюднено на X Міжнародній науково-теоретичній конференції “Modernization of today’s science: experience and trends” (м. Глазго, Велика Британія, 22 травня 2026 року). За матеріалами конференції опубліковано тези доповіді [22]; копію публікації наведено в додатку Б.

Структура роботи. Робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел і додатків. У першому розділі проаналізовано прикладні задачі оптимізації та наявні програмні засоби, визначено постановку задачі розробки й цільову аудиторію системи. У другому розділі

розглянуто методи розв’язання п’яти класів задач і обґрунтовано вибір технологій реалізації. У третьому розділі описано програмну реалізацію системи: серверну частину, сховище даних, REST API та клієнтський застосунок. У четвертому розділі викладено роботу користувача з системою та окреслено перспективи її розвитку. Два додатки доповнюють роботу: у додатку А наведено програмний код ключових обчислювальних модулів системи, а в додатку Б — копію публікації тез доповіді за темою дослідження. Найбільшу за обсягом частину займає другий розділ, що відповідає його ключовій ролі — теоретичному обґрунтуванню методів, на яких ґрунтується вся система.

1 АНАЛІЗ ПРИКЛАДНИХ ЗАДАЧ ОПТИМІЗАЦІЇ ТА ОГЛЯД ПРОГРАМНИХ ЗАСОБІВ ЇХ РОЗВ'ЯЗАННЯ

Розв'язання прикладної оптимізаційної задачі спирається водночас на дві передумови: коректну математичну постановку самої задачі та наявність доступного програмного засобу, здатного цю постановку обробити й подати результат у зрозумілій формі. Тому, перш ніж формулювати вимоги до власної системи підтримки прийняття рішень, доцільно окреслити коло прикладних задач, які вона має охоплювати, і проаналізувати наявні інструменти їх розв'язання — з'ясувати їхні можливості, обмеження та неперекриті ніші.

1.1 Постановка задачі

Метою розробки є створення вебсистеми підтримки прийняття рішень, що надає особі, яка ухвалює рішення, інтегрований інструмент формулювання й автоматизованого розв'язання типових задач математичного програмування п'яти класів у межах єдиного браузерного інтерфейсу, без потреби встановлювати спеціалізоване програмне забезпечення чи володіти алгебраїчними мовами опису оптимізаційних моделей. Прикладною задачею, на розв'язанні якої апробовано систему, є щоденне планування й перерозподіл обмежених ресурсів у плановій, виробничій і логістичній діяльності. У таких умовах управлінець постає перед необхідністю обрати з-поміж скінченної кількості допустимих варіантів той, що забезпечує найкраще значення обраного критерію — мінімум витрат, максимум прибутку чи мінімум сумарного часу — за дотримання сукупності балансових, ресурсних і технологічних обмежень.

Розрахунковою аудиторією системи є дві категорії користувачів. Перша — фахівці-практики (планові, логістичні та аналітичні підрозділи підприємств

торговельно-виробничого профілю), для яких розподільні задачі виникають як регулярна виробнича рутина, проте опанування промислового математичного інструментарію виходить за межі їхньої фахової підготовки. Друга — студенти й викладачі технічних спеціальностей, для яких система слугує навчальною платформою, що дозволяє простежити поведінку класичних оптимізаційних алгоритмів на конкретних прикладах і зіставити результати застосування різних методів до однакових вихідних даних.

Розроблювана система охоплює п’ять класів оптимізаційних задач, які покривають типові управлінські сценарії: задачу лінійного програмування з неперервними змінними (розподіл подільних ресурсів між альтернативними варіантами використання); задачу цілочислового та змішаного лінійного програмування (планування з неподільними одиницями — кількість одиниць продукції, бінарні рішення “виконувати/не виконувати”); транспортну задачу в класичній постановці (мінімізація сумарної вартості перевезень від групи постачальників до групи споживачів); задачу про призначення (закріплення виконавців за роботами з найкращим сумарним показником ефективності — у класичній постановці n×n або довільній прямокутній m×n зі зведенням до квадратної форми); задачу про рюкзак у формулюванні 0/1 (відбір підмножини елементів обмеженої сумарної ваги з максимальною сумарною цінністю). Кожному з перелічених класів відповідає історично усталений метод розв’язання, що використовує специфічну структуру задачі для отримання обчислювальної переваги перед загальними підходами.

Постановку задачі користувач формує одним із двох способів. Перший — заповнення спеціалізованої шаблонної форми, в якій структура полів повторює математичну специфіку відповідного класу задачі (вектор пропозицій та вектор попиту разом із матрицею тарифів для транспортної задачі; перелік предметів із вагами і цінностями та обмеження місткості для задачі про рюкзак тощо). Другий — вільне формулювання задачі лінійного програмування через перелік змінних із зазначенням типу й допустимих меж, коефіцієнтів цільової функції з вказівкою

напрямку оптимізації та системи лінійних обмежень. Незалежно від обраного способу формулювання сервіс класифікації автоматично визначає тип задачі за її структурою і пропонує метод розв’язання з текстовим обґрунтуванням обраного варіанту, причому користувач за потреби перевизначає рекомендований метод явно.

Окрім самого розв’язку, система надає три додаткові інформаційні шари, які роблять процес обчислень прозорим для користувача без фахової підготовки в галузі дослідження операцій. Перший — покрокова візуалізація ітерацій методу, специфічна для кожного класу задачі (симплекс-таблиця, дерево гілкування, матриця алокацій із циклом перерозподілу, матриця з лініями покриття, таблиця динамічного програмування). Другий — графічна побудова допустимої області для лінійних задач у дво- і тривимірному просторі. Третій — аналіз чутливості оптимального розв’язку до зміни коефіцієнтів цільової функції та правих частин обмежень для задач лінійного програмування з неперервними змінними. Усі виконані розв’язання зберігаються в персональній історії, прив’язаній до облікового запису користувача, з можливістю фільтрації, клонування, формування what-if-варіантів зі зміненими параметрами та генерації PDF-звіту.

Таким чином, задачу розробки формулюємо так: створити єдину веборієнтовану систему підтримки прийняття рішень, що охоплює всі п’ять класів задач і поєднує автоматичне розв’язання з прозорою для непідготовленого користувача покроковою демонстрацією процесу обчислень.

1.2 Аналіз існуючих програмних засобів

Перед розробкою інтегрованої системи підтримки прийняття рішень доцільно проаналізувати наявні на ринку програмні засоби, що дозволяють формулювати й розв’язувати задачі математичного програмування, з метою з’ясування їхніх переваг, обмежень і неперекритих ніш. Існуючі рішення можна

систематизувати за двома незалежними ознаками: рівнем фахової підготовки, потрібним користувачеві для ефективної роботи з продуктом, і ступенем прозорості процесу обчислень — здатністю інтерфейсу простежити проміжні стани алгоритму, а не лише видавати кінцевий результат. Розглянемо характерні категорії існуючих рішень.

Найбільш потужний клас засобів організовано у двошарову архітектуру: верхній рівень становлять інструменти моделювання, які дають змогу описувати задачу мовою, наближеною до математичної нотації, нижній — обчислювальні рушії (солвери), що виконують власне процедуру оптимізації. До комерційних інструментів моделювання належать алгебраїчні мови AMPL, GAMS та AIMMS [1, 2, 3], які мають високу виразну спроможність, проте поширюються за платними ліцензіями зі значною вартістю, надаючи безкоштовний доступ лише для академічних установ. У зв'язку з цими мовами використовуються комерційні солвери Gurobi та CPLEX [4, 5], які демонструють найкращі показники швидкодії на задачах змішаного цілочислового програмування великої розмірності. Спільним обмеженням комерційного двошарового стека є висока вимога до підготовки користувача: ефективна робота передбачає володіння теорією дослідження операцій і навичками алгебраїчного запису задач — високий поріг входження окремо відзначено у тематичних оглядах.

Відкритою альтернативою комерційним пакетам є бібліотеки моделювання Pyomo та PuLP для мови Python та JuMP для мови Julia [6, 7, 8], які знімають фінансовий бар'єр, проте перекладають на користувача необхідність володіти базовою програмістською підготовкою. У зв'язку з ними використовуються відкриті солвери CBC, GLPK та HiGHS [9]; окрему позицію посідає інструментарій OR-Tools від Google, який поєднує власний набір солверів — зокрема CP-SAT для задач з обмеженнями — із вбудованими засобами моделювання. Прозорість процесу обчислень у двошаровому стеці — як комерційному, так і відкритому — залишається мінімальною: стандартний робочий сценарій передбачає одержання кінцевого розв'язку без перегляду проміжних ітерацій.

Окрему групу формує вбудований модуль “Пошук рішення” (Solver) табличного процесора Microsoft Excel [10], який дозволяє формулювати задачі лінійного, цілочислового та нелінійного програмування у вигляді клітинок робочого аркуша й використовує комерційний солвер фірми Frontline Systems. Перевагами підходу є інтеграція з електронними таблицями, де часто вже зберігаються вихідні дані, і відносна простота освоєння для офісного користувача. Серед обмежень — відсутність прозорості процесу обчислень (результат подається без розгортання проміжних кроків симплекс-методу) і неявне ведення історії розв’язань через копіювання аркушів. Аналіз чутливості представлений у вигляді окремого Sensitivity Report, що генерується додатковим викликом і подається текстовою таблицею з тіньовими цінами та діапазонами допустимості; інтерактивної графічної інтерпретації (наприклад, у вигляді tornado-діаграми тіньових цін) Excel Solver не передбачає.

Окремий клас становлять веб-калькулятори окремих методів — PHPSimplex, simplex.online та подібні платформи, що реалізують симплекс-метод, метод потенціалів або угорський алгоритм у формі простої табличної форми введення [11]. Користувач уводить коефіцієнти задачі і одержує числовий розв’язок без необхідності встановлювати програмне забезпечення чи володіти алгебраїчним записом; частина калькуляторів додатково відображає симплекс-таблиці кожної ітерації, що робить їх корисними у навчальному контексті. Спільним обмеженням цього класу інструментів є тематична вузькість: кожен калькулятор реалізує один метод для одного класу задач, не зв’язує постановку й результат у єдину історію роботи користувача, не передбачає аналізу чутливості й не надає засобів повернення до попереднього формулювання для зміни параметрів.

Близькі за функціональністю — навчальні середовища, орієнтовані на викладання теорії оптимізації: інтерактивні модулі Wolfram Demonstrations, навчальні аплети курсів edX і Coursera, спеціалізовані веб-додатки, розроблені викладачами окремих дисциплін [12]. Спільною рисою таких систем є висока прозорість процесу обчислень — користувач спостерігає кожен крок алгоритму з

поясненнями — у поєднанні з вузькою тематичною прив’язкою до конкретного навчального матеріалу: один аплет демонструє симплекс-метод на двовимірних прикладах, інший — метод гілок і меж на дереві фіксованої глибини, третій — задачу про рюкзак малої розмірності. Жоден з розглянутих навчальних інструментів не пропонує єдиного інтерфейсу для задач різних класів і, як правило, не передбачає збереження персональної історії розв’язань.

У порівнянні з переліченими рішеннями розроблювана вебсистема формується як спроба заповнити неперекриту попередніми класами комбінацію вимог. Від навчальних середовищ вона успадковує орієнтацію на користувача без фахової підготовки в галузі дослідження операцій і покрокову візуалізацію процесу обчислень; від двошарових платформ моделювання — широту охоплення (підтримка п’яти класів задач у межах одного застосунку); від онлайн-калькуляторів — простоту веб-доступу без необхідності встановлення програмного забезпечення. На відміну від кожного з перелічених класів окремо, система додає функціональні можливості, відсутні в існуючих рішеннях у такому поєднанні: автоматичну класифікацію типу задачі і рекомендацію методу з обґрунтуванням, аналіз чутливості для задач лінійного програмування, ведення персональної історії розв’язань з прив’язкою до облікового запису та підтримку what-if-варіантів зі збереженням посилання на базову задачу.

2 МЕТОДИ ОПТИМІЗАЦІЇ ТА ОБРАНІ ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ

Розділ містить теоретичний опис п’яти методів математичного програмування, обраних у підрозділі 1.1 для реалізації в обчислювальному ядрі розробленої системи: симплекс-методу для задач лінійного програмування з неперервними змінними, методу гілок і меж для змішаного цілочислового лінійного програмування (ЦЛП), методу потенціалів для класичної транспортної задачі, угорського алгоритму для задачі про призначення та динамічного програмування для задачі про рюкзак у формулюванні 0/1. Для кожного методу наведено формальну постановку відповідної задачі, ідею алгоритму та опис ключових кроків — за допомогою математичних формул або псевдокоду, залежно від того, яка форма подання надає краще уявлення про специфіку методу. Алгоритмічні деталі реалізації — структури даних, обробка особливих випадків, взаємодія між модулями обчислювального ядра — винесено в розділ 3. У підрозділі 2.6 наведено зіставлення методів за класами задач, обчислювальною складністю та гарантіями оптимальності, а підрозділ 2.7 обґрунтовує вибір технологій реалізації системи.

2.1 Симплекс-метод

Симплекс-метод, запропонований Дж. Данцигом у 1947 році, є базовим обчислювальним алгоритмом розв’язання задач лінійного програмування з неперервними змінними [13, 14]. Незважаючи на більш ніж сімдесятирічну історію, метод залишається стандартом галузі — як з огляду на обчислювальну ефективність на практичних задачах, так і через природне поєднання з апаратом аналізу чутливості.

Класична задача лінійного програмування у стандартній формі формулюється у вигляді (2.1):

$$z = c^T x \rightarrow max,\ A\,x \leq b,\ x \geq 0 \qquad (2.1)$$

де $x = (x_1, x_2, \ldots, x_n)^T$ — вектор шуканих змінних;

c — вектор коефіцієнтів цільової функції;

A — матриця коефіцієнтів обмежень розмірності m × n;

b — вектор правих частин обмежень.

Задача мінімізації зводиться до задачі максимізації через інверсію знаків коефіцієнтів цільової функції.

Геометричною інтерпретацією задачі є пошук точки багатогранника допустимих розв'язків, у якій лінія рівня цільової функції набуває найбільшого значення. Відомо, що якщо така точка існує, вона є хоча б однією з вершин багатогранника, а отже замість континуального простору достатньо розглядати скінченну множину вершин. Саме на цій геометричній властивості й базується ідея методу: послідовний перехід між сусідніми вершинами в напрямку покращення цільової функції до досягнення оптимальної вершини або встановлення факту необмеженості розв'язку.

Алгоритмічно перехід між вершинами реалізується через перетворення симплекс-таблиці — таблиці коефіцієнтів задачі, доповненої штучними та допоміжними змінними для приведення обмежень до канонічної форми. Одна ітерація методу полягає у виборі небазисної змінної з від'ємним коефіцієнтом у рядку цільової функції (та, що увійде до базису; конкретне правило вибору з-поміж кількох кандидатів описано нижче), визначенні базисної змінної, яку слід вивести з базису (за правилом мінімального відношення b_i до коефіцієнта розв'язувального стовпця) та перерахунку таблиці методом виключення Гауса-Жордана. Процес продовжується доти, доки всі коефіцієнти в рядку цільової функції не стануть невід'ємними — у такому разі досягнуто оптимальної вершини.

Для задач, у яких система обмежень не дозволяє очевидно вибрати початковий базис (зокрема, коли наявні обмеження зі знаками "≥" або "="),

використовується техніка Big-M [15]. Її ідея полягає у введенні штучних змінних з великим штрафом M у цільовій функції — це гарантує, що в оптимальному розв'язку всі штучні змінні наберуть нульового значення, а їхня присутність у початковому базисі автоматизує процедуру побудови першого допустимого плану. Значення M обирається настільки великим, щоб домінувати над будь-яким реальним коефіцієнтом цільової функції; у програмній реалізації M фіксується константою порядку 10^6, що достатньо для всіх практичних розмірностей задач, які підтримує система.

На вироджених задачах — таких, у яких на одній ітерації відразу кілька змінних претендують на вихід з базису через однакові мінімальні відношення — теоретично можливе явище циклічності: послідовність ітерацій повертається до однієї й тієї самої вершини, не покращуючи значення цільової функції. Для гарантованого запобігання циклічності у програмній реалізації на кожній ітерації застосовано правило Бленда [16]: серед кандидатів на введення до базису обирається змінна з найменшим індексом, серед кандидатів на виведення — також змінна з найменшим індексом. Це правило не є найшвидшим за середньою кількістю ітерацій, проте забезпечує скінченність алгоритму на будь-якій вхідній задачі.

Обчислювальна складність симплекс-методу у гіршому випадку є експоненційною від кількості змінних — відома задача Клее-Мінті, на якій метод виконує 2^n ітерацій для n-вимірного куба [14]. Проте на типових прикладних задачах метод демонструє практично поліноміальну поведінку, виконуючи близько $O(m)$ або $O(m \cdot n)$ ітерацій. Саме ця практична властивість, у поєднанні з природним отриманням базисної інформації наприкінці виконання (потрібної для аналізу чутливості), зробила симплекс-метод стандартом галузі попри теоретичну експоненційну складність. Крім самостійного застосування до задач лінійного програмування з неперервними змінними, симплекс-метод слугує обчислювальною основою для методів, які на кожному кроці розв'язують лінійну релаксацію вихідної задачі, — зокрема для розглянутого далі методу гілок і меж.

2.2 Метод гілок і меж

Метод гілок і меж у класичному вигляді запропонували А. Г. Доіг та А. Х. Ленд у 1960 році для розв’язання задач ЦЛП та змішаного ЦЛП — таких, у яких частина або всі змінні обмежені умовою цілочисловості [17]. Безпосереднє застосування симплекс-методу до подібних задач у загальному випадку неможливе: оптимум у вершині багатогранника, отриманий симплексом, не зобов’язаний мати цілі координати, а звичайне округлення дробових значень не гарантує ані оптимальності, ані навіть допустимості одержаного цілочислового розв’язку.

Ідея методу полягає у систематичному розбитті початкової задачі на сімейство підзадач, кожна з яких більш обмежена, ніж попередня. Розбиття починається з так званої LP-релаксації — ослабленої версії задачі, у якій умови цілочисловості тимчасово вилучено й залишено лише вимоги до знаків та меж змінних. Розв’язок LP-релаксації, обчислений симплекс-методом, дає верхню оцінку оптимуму для задачі максимізації або нижню — для задачі мінімізації. Якщо в цьому розв’язку всі цілочислові змінні набули цілих значень, задачу розв’язано; інакше серед них обирається змінна для гілкування і породжуються дві дочірні підзадачі, що відсікають дробове значення обраної змінної.

Правилом гілкування у розробленій системі обрано вибір найбільш дробової змінної — тієї, дробова частина якої найближча до 0,5. На обрану змінну x_j з поточним дробовим значенням x_j* у дочірніх підзадачах додаються взаємовиключні обмеження (2.2):

$$x_j \leq \lfloor x_j^* \rfloor \quad \text{або} \quad x_j \geq \lceil x_j^* \rceil \tag{2.2}$$

Кожна з цих підзадач знов розв’язується LP-релаксацією і породжує власні дочірні вузли — таким чином формується дерево вузлів, корінь якого відповідає початковій задачі, а листи — підзадачам, які або дають цілочисловий розв’язок, або є несумісними, або відсікаються через невигідну оцінку.

Стратегією обходу дерева в розробленій системі обрано best-first: на кожному

кроці з пріоритетної черги витягається вузол з найкращим (за напрямком оптимізації) значенням LP-релаксації. Така стратегія дозволяє раніше досягати високоякісних цілочислових розв'язків і ефективно відсікати малоперспективні гілки. Загальну схему алгоритму наведено на рисунку 2.1.

```
PriorityQueue Q ← { LP relaxation of the initial problem }
incumbent ← ∅;   bound ← -∞                  // for maximization

while Q ≠ ∅ do
    node ← extract node with the best LP-bound from Q
    if LP-bound(node) ≤ bound then continue          // prune by bound
    if LP(node) is infeasible then continue         // prune by infeasibility
    if all integer variables in LP(node) are integral then
        if obj(node) > bound then
            incumbent ← node;   bound ← obj(node)
        continue
    var ← most fractional variable in LP(node)
    Q.push( node with constraint  var ≤ ⌊var*⌋ )
    Q.push( node with constraint  var ≥ ⌈var*⌉ )
end while

return incumbent
```

Рисунок 2.1 — Псевдокод методу гілок і меж

Вузол відсікається у трьох випадках: коли LP-релаксація встановлює несумісність підзадачі, коли LP-оцінка вузла гірша за поточне найкраще цілочислове значення (incumbent), а також коли всі цілочислові змінні в LP-розв'язку вже мають цілі значення (вузол стає кандидатом на нового incumbent).

У гіршому випадку дерево вузлів зростає експоненційно від кількості цілочислових змінних — 2^k вершин для k бінарних змінних. Проте на практиці завдяки правильно обраній стратегії гілкування і відсіченню більшість гілок не досліджується до кінця, що робить метод придатним для задач навчально-академічного масштабу, на які орієнтована розроблена система. Для контролю над патологічними випадками програмна реалізація обмежує максимальну кількість досліджуваних вузлів значенням 1000 — при перевищенні цього порога повертається повідомлення про помилку з ясним текстом без зависання процесу.

2.3 Метод потенціалів

Метод потенціалів (MODI, modified distribution method) є спеціалізованим алгоритмом розв'язання класичної транспортної задачі [18]. Хоча транспортну задачу можна формально звести до загальної задачі лінійного програмування й розв'язати симплекс-методом, її блочна структура — обмеження мають вигляд балансів за рядками й стовпцями — дозволяє побудувати значно ефективніший алгоритм, що працює безпосередньо з матрицею перевезень, без розгортання у плоску постановку лінійного програмування.

Класична постановка транспортної задачі описує систему з m постачальників, що мають обсяги пропозиції $a_1, \ldots, a_m$, і n споживачів з обсягами попиту $b_1, \ldots, b_n$. Задано матрицю одиничних тарифів c_ij — вартість перевезення одиниці продукції від постачальника i до споживача j. Потрібно знайти план перевезень x_ij — кількість продукції, що перевозиться між кожною парою "постачальник-споживач", — який мінімізує сумарну вартість при дотриманні балансових обмежень (2.3):

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij}$$

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

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

$$x_{ij} \geq 0 \tag{2.3}$$

де c_{ij} — вартість перевезення одиниці продукції від постачальника i до споживача j;

x_{ij} — обсяг перевезення від постачальника i до споживача j;

a_i — обсяг пропозиції постачальника i, i = 1, …, m;

b_j — обсяг попиту споживача j, j = 1, …, n.

Передбачається збалансованість задачі: $\sum a_i = \sum b_j$. У разі незбалансованості — коли сумарна пропозиція не дорівнює сумарному попиту — задачу попередньо балансують введенням фіктивного постачальника або споживача з нульовими тарифами та обсягом, що компенсує дисбаланс; на оптимальний план фіктивні рядок або стовпець не впливають, лише поглинають надлишок.

Початковий опорний план будується методом північно-західного кута: послідовно заповнюються клітинки матриці перевезень, починаючи з верхнього лівого кута, мінімальним значенням з пари (залишок пропозиції рядка, залишок попиту стовпця). Така процедура гарантує побудову плану з рівно (m + n − 1) ненульовими клітинами — це необхідна умова невиродженості, що дозволяє однозначно обчислити потенціали наступного кроку.

На кожній ітерації алгоритму обчислюються потенціали рядків і стовпців u_i та v_j з умови, що для кожної базисної клітини (i, j) виконується рівність (2.4):

$$u_i + v_j = c_{ij}, \quad (i,j) — \text{базисна клітина}$$
$$\Delta_{ij} = c_{ij} - u_i - v_j, (i,j) — \text{небазисна клітина} \qquad (2.4)$$

де u_i — потенціал рядка i, i = 1, …, m;

v_j — потенціал стовпця j, j = 1, …, n;

c_ij — тариф перевезення одиниці продукції від постачальника i до споживача j;

Δ_ij — зведена вартість небазисної клітини (i, j).

Оскільки система рівнянь для базисних клітин містить (m + n − 1) рівнянь і (m + n) невідомих, один з потенціалів обирається довільно (зазвичай $u_1 = 0$), а решта обчислюються послідовним розкриттям системи. Якщо для всіх небазисних клітин виконується умова $\Delta_{ij} \geq 0$ — поточний план оптимальний. Якщо ж існує клітина (i*, j*) зі значенням Δ_(i*j*) < 0, відповідна змінна перспективна для введення до базису, і алгоритм виконує її введення з одночасним виведенням однієї з базисних змінних.

Введення нової клітини у базис супроводжується перерозподілом по замкненому циклу: знаходиться замкнений цикл, який починається і закінчується у

клітині (i*, j*) та проходить лише через базисні клітини з почерговою зміною напрямку “горизонталь-вертикаль”. Уздовж циклу клітини позначаються знаками “+” і “–”, починаючи з “+” у клітині, що вводиться. Мінімальне значення серед клітин зі знаком “–” становить величину θ, на яку коригується план: до клітин з “+” додається θ, від клітин з “–” — віднімається θ. Клітина з “–”, на якій досягається мінімум, виходить з базису; нова клітина займає її місце. Алгоритм повторюється до досягнення оптимальності всіх зведених вартостей.

У вироджених випадках, коли в результаті перерозподілу одночасно кілька клітин обнуляються (величина θ досягається на двох або більше клітинах одночасно), для збереження зв’язності базису й коректного обчислення потенціалів на наступній ітерації вводиться нульова алокація — одна з обнулених клітин примусово залишається в базисі зі значенням нуль. Така технічна процедура не впливає на чисельний результат, але запобігає виродженню системи рівнянь для потенціалів і гарантує коректне продовження алгоритму.

2.4 Угорський алгоритм

Угорський алгоритм, запропонований Г. Куном у 1955 році на основі робіт угорських математиків Д. Кеніга й Є. Егерварі, розв’язує задачу про призначення — задачу віднесення виконавців до робіт за умови мінімізації сумарної вартості (або максимізації сумарного прибутку), де кожен виконавець призначається рівно на одну роботу і кожна робота — рівно одному виконавцеві [19]. Хоча задачу можна формально звести до транспортної (вироджений випадок з одиничними обсягами пропозиції й попиту) або до загального лінійного програмування, її особлива структура — бінарна матриця призначень з обмеженнями типу “рівно одна одиниця в кожному рядку і кожному стовпці” — дозволяє побудувати алгоритм поліноміальної складності, що працює безпосередньо з матрицею вартостей.

Алгоритм оперує квадратною матрицею C розмірності n × n, у якій елемент c_{ij} позначає вартість виконання роботи j виконавцем i. Якщо вхідна матриця прямокутна, тобто кількість виконавців не дорівнює кількості робіт, вона автоматично доповнюється фіктивними рядками або стовпцями з нульовими елементами до квадратної форми — це не впливає на оптимальний розв'язок, оскільки фіктивні призначення мають нульову вартість і не змінюють сумарного значення. Задача максимізації зводиться до задачі мінімізації шляхом інверсії елементів матриці: кожен елемент замінюється на різницю між максимумом матриці і самим елементом.

Формальна постановка задачі про призначення має такий вигляд. Дано n виконавців і n робіт; кожен виконавець i може бути призначений на будь-яку роботу j з відомою вартістю c_{ij}. Введемо бінарну змінну x_{ij}, що дорівнює 1, якщо виконавець i призначений на роботу j, і 0 — у протилежному випадку. Тоді задача мінімізації сумарної вартості формулюється у вигляді (2.5):

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

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \dots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \dots, n$$

$$x_{ij} \in \{0, 1\} \tag{2.5}$$

де c_{ij} — вартість виконання роботи j виконавцем i;

x_{ij} — індикатор призначення виконавця i на роботу j;

n — кількість виконавців і робіт (після доповнення матриці до квадратної форми).

Перше обмеження гарантує, що кожен виконавець отримує рівно одну роботу, друге — що кожна робота закріплюється рівно за одним виконавцем; вимога бінарності змінних робить задачу формально цілочисловою, проте завдяки

властивостям повної унімодулярності матриці обмежень її розв’язок збігається з розв’язком неперервної релаксації, тобто може бути знайдений за поліноміальний час.

Класичний матричний варіант алгоритму Куна-Мункреса послідовно виконує чотири кроки. Спочатку у кожному рядку матриці знаходиться мінімальний елемент і віднімається від усіх елементів цього рядка — після цього в кожному рядку гарантовано присутній принаймні один нуль. Далі ту саму процедуру виконують за стовпцями: у кожному стовпці одержаної матриці знаходиться мінімальний елемент і віднімається від усіх елементів цього стовпця, після чого нуль присутній і в кожному рядку, і в кожному стовпці. На третьому кроці виконується пошук максимального парування на двочастковому графі, вершинами якого є рядки і стовпці матриці, а ребрами — позиції нульових елементів; парування шукається обходом у глибину з послідовним розкриттям нульових клітин і перепризначенням попередніх збігів, і якщо одержано парування розміру n — задача розв’язана, знайдена сукупність нульових клітин визначає оптимальне призначення виконавців на роботи. Якщо ж парування неповне (розмір менший за n), на завершальному кроці будується мінімальне покриття нульових клітин горизонтальними і вертикальними лініями за теоремою Кеніга: серед усіх непокритих клітин знаходиться мінімальний елемент δ, він віднімається від усіх непокритих рядків і додається до всіх покритих стовпців — у непокритих ділянках матриці з’являються нові нулі, що дозволяє знайти більше парування на наступній ітерації, не порушуючи досягнутих властивостей матриці.

Кроки 3 і 4 повторюються до досягнення повного парування. Загальна обчислювальна складність алгоритму становить $O(n^3)$, що робить його значно ефективнішим за пряму редукцію до лінійного програмування. Поліноміальність у поєднанні з лаконічним матричним представленням з лініями покриття і нульовими клітинами роблять угорський алгоритм найприроднішим інструментом для класу задач про призначення — як з обчислювальних, так і з демонстраційних міркувань.

2.5 Динамічне програмування

Динамічне програмування як загальна методологія оптимізації запропоноване Р. Беллманом у 1950-х роках і ґрунтується на принципі оптимальності: оптимальний розв'язок задачі складається з оптимальних розв'язків її підзадач [20]. У розробленій системі цю методологію застосовано для класу задач про рюкзак у формулюванні 0/1, де вимагається обрати таку підмножину предметів із заданими цілими вагами і цінностями, щоб сумарна цінність набору була максимальною за дотримання обмеження на сумарну вагу.

Формальна постановка задачі про рюкзак така. Задано n предметів; кожен предмет і має цілу вагу w_i і цінність v_i. Задано місткість рюкзака W. Вводиться бінарна змінна $x_i \in \{0, 1\}$, яка дорівнює 1, якщо предмет і включено до набору, і 0 — у протилежному випадку. Потрібно знайти набір, що максимізує сумарну цінність $\sum v_i x_i$ при дотриманні обмеження на вагу $\sum w_i x_i \leq W$.

Принцип оптимальності для цієї задачі реалізується через побудову двовимірної таблиці f[i][w], елемент якої відповідає максимальній сумарній цінності набору, складеного з перших і предметів, за умови що сумарна вага не перевищує w. Для кожної клітини маємо дві альтернативи: або предмет і не входить до набору (тоді результат збігається з f[i−1][w]), або входить, за умови $w_i \leq w$ (тоді результат становить $f[i-1][w - w_i] + v_i$). З двох альтернатив обирається більша. Це твердження записується рекурентним співвідношенням Беллмана (2.6):

$$f[i][w] = max(f[i-1][w], \quad f[i-1][w - w_i] + v_i), \quad \text{якщо } w_i \leq w$$
$$f[i][w] = f[i-1][w], \quad \text{якщо } w_i > w$$
$$f[0][w] = 0, \quad \text{для всіх } w = 0, 1, \ldots, W \qquad (2.6)$$

де f[i][w] — максимальна сумарна цінність набору з перших і предметів за обмеженням ваги w;

w_i — вага предмета i;

v_i — цінність предмета i;

W — місткість рюкзака.

Таблиця заповнюється послідовно за зростанням i (від 1 до n) і за зростанням w (від 0 до W); після заповнення значення f[n][W] становить шукану максимальну сумарну цінність. Сам набір обраних предметів відновлюється зворотним проходом по таблиці від останнього предмета до першого; відповідний алгоритм наведено на рисунку 2.2.

```
items ← ∅;   i ← n;   w ← W
while i > 0 do
    if f[i][w] ≠ f[i−1][w] then // item i is in the optimal subset
        items.add(i)
        w ← w − w_i
    i ← i − 1
return items
```

Рисунок 2.2 — Псевдокод відновлення оптимального набору предметів

Логіка зворотного проходу полягає у порівнянні f[i][w] зі значенням, яке було б у разі відкидання i-го предмета (f[i−1][w]): якщо вони різняться — предмет було включено в оптимальний набір; якщо збігаються — не було. Після виходу з циклу набір items містить індекси всіх обраних предметів.

Обчислювальна складність алгоритму становить O(n·W), де n — кількість предметів, W — місткість рюкзака. Така складність називається псевдополіноміальною: вона поліноміальна за числовими значеннями вхідних даних, проте експоненційна відносно довжини їхнього бінарного запису. На задачах академічного й помірного прикладного масштабу, де W не перевищує кількох тисяч одиниць, метод залишається практично прийнятним, що відповідає цільовому призначенню розробленої системи. Ключова перевага підходу — гарантія точного оптимуму: на відміну від наближених жадібних евристик, динамічне програмування не “застрягає” в локально вигідному, але глобально неоптимальному виборі, оскільки спирається на принцип оптимальної підструктури. Це й зумовило вибір методу для задачі про рюкзак, де потрібен

гарантовано найкращий, а не приблизний набір предметів.

2.6 Порівняльний аналіз методів за класами задач

У підрозділах 2.1–2.5 описано п’ять методів оптимізації, кожен з яких реалізує специфічний клас задач математичного програмування. У цьому підрозділі наведено зведене порівняння методів за чотирма ключовими характеристиками: клас задач, до якого метод застосовується; обчислювальна складність у гіршому випадку і на типових задачах; альтернативні підходи, які розглядалися, але були відхилені на користь обраного методу. Зведену характеристику наведено в таблиці 2.1.

Таблиця 2.1 — Порівняльна характеристика реалізованих методів

Метод	Клас задач	Складність	Чому обрано саме цей
Симплекс-метод	ЛП, неперервні змінні	O(m·n) типово, експ. у гіршому	базисна інформація для аналізу чутливості, наочне геометричне тлумачення
Метод гілок і меж	змішане ЦЛП	експ., залежить від задачі	дерево вузлів інтуїтивніше за відсічення Гоморі
Метод потенціалів (MODI)	транспортна задача	O(m·n·(m+n)) на ітерацію	зберігає блочну структуру, прозорий план перевезень
Угорський алгоритм	задача про призначення	$O(n^3)$	пряма робота з матрицею, поліноміальна складність
Динамічне програмування	задача про рюкзак 0/1	O(n·W) псевдополін.	таблиця прозоро відображає принцип Беллмана

Усі п’ять реалізованих методів є точними: при коректному застосуванні вони гарантовано знаходять глобальний оптимум, якщо той існує, або встановлюють факт необмеженості чи несумісності задачі. Це принципово відрізняє розроблену систему від інструментів, що використовують евристичні методи (генетичні

алгоритми, імітація відпалу тощо): користувач отримує не наближене, а точне розв’язання задачі — що важливо для прикладних сценаріїв, у яких управлінське рішення приймається на підставі одержаного оптимуму.

Спільною властивістю обраних методів є придатність до покрокової візуалізації, специфічної для кожного класу задачі. Саме поєднання точності, прийнятної обчислювальної складності на навчально-академічних розмірностях і наочної візуалізації визначило склад обчислювального ядра розробленої системи.

Окремо слід відзначити, що симплекс-метод посідає центральне місце в обчислювальному ядрі. Окрім самостійного розв’язання задач лінійного програмування з неперервними змінними, він використовується як LP-розв’язувач у методі гілок і меж для обчислення релаксацій вузлів дерева. Така архітектура дозволяє повторно використовувати єдину реалізацію алгоритму, забезпечуючи узгодженість поведінки обох солверів і спрощуючи супровід обчислювального ядра.

2.7 Технології реалізації

Основою серверної частини обрано мову Python 3.11 — заради зрілої екосистеми наукових бібліотек і спільної мови з обчислювальним ядром. Вебсервіс будовано на фреймворку FastAPI з бібліотекою Pydantic v2, обраних за декларативний опис і автоматичну типізовану валідацію контрактів REST API. Доступ до реляційної бази — SQLAlchemy 2 з міграціями Alembic, керування залежностями — uv. Допоміжні задачі покрито перевіреними бібліотеками з відкритим кодом: JWT-автентифікація з хешуванням паролів, генерація PDF-звітів (ReportLab) і побудова графіків до них (matplotlib).

Розмічені об’єднання (discriminated unions) Pydantic v2 обрано тому, що вони дають змогу описати всі п’ять класів задач єдиним контрактом і формально верифікувати запит ще до обчислень — за значенням спеціального поля-

дискримінатора бібліотека сама добирає потрібну схему валідації. Варто розрізняти п’ять класів постановок задач і п’ять обчислювальних методів ядра: ці переліки не збігаються поелементно — задачу планування заряду-розряду акумулятора зведено до лінійної моделі й делеговано симплекс-методу, а цілочислові лінійні задачі описуються тією самою схемою, що й неперервні, але розв’язуються методом гілок і меж.

Версію 2 бібліотеки SQLAlchemy обрано через підтримку типізованих атрибутів моделей: у поєднанні з суворим режимом TypeScript на клієнті це забезпечує наскрізну типобезпеку контрактів — від форми введення задачі у браузері до запису в таблицю бази даних на сервері.

Обчислювальне ядро побудовано на бібліотеці NumPy — фактичному стандарті наукових обчислень мовою Python; усі п’ять методів реалізовано самостійно на матрицях NumPy, без готових солверів. Така принципова відмова від готових солверів (SciPy, PuLP, OR-Tools) зумовлена вимогою покрокової візуалізації: вони повертають лише кінцевий розв’язок і не дають доступу до внутрішнього стану алгоритму на проміжних ітераціях, а спроба перехопити їхні внутрішні виклики була б крихкою й залежною від недокументованих інтерфейсів. Самостійна реалізація дає повний контроль над станом кожної ітерації; SciPy задіяно лише в тестах як еталон крос-верифікації й відсутня в продуктивних залежностях, що підтверджує самостійність реалізації методів.

Для клієнтської частини обрано стек Next.js 15 з React 19 і TypeScript. Стилізація — Tailwind CSS з компонентною бібліотекою shadcn/ui на примітивах Radix UI. Управління формами — react-hook-form і Zod; HTTP-запити до сервера — axios; кешування відповідей — TanStack Query. Покрокові візуалізації — Plotly.js (графіки допустимої області), React Flow (дерево гілкування), KaTeX (формули).

Модель App Router у Next.js 15 обрано за можливість поєднати серверний і клієнтський рендеринг в одному застосунку, що раціонально розподіляє навантаження між сервером і браузером.

Суворий режим TypeScript на клієнті обрано заради наскрізної типобезпеки: типи структур задач узгоджено з контрактами сервера, тож помилки невідповідності виявляються ще під час компіляції, а не у браузері.

На рівні даних — СКБД PostgreSQL 16 з підтримкою JSONB для зберігання формулювань задач різних типів в одному полі. Розгортання у контейнері Docker.

Єдине поле типу JSONB для всіх п’яти класів задач обрано як альтернативу окремій таблиці на кожен тип: додавання нового типу задачі тоді не потребує міграції схеми бази, а зводиться до опису нової Pydantic-схеми в коді. Цілісність документа в цьому полі гарантується валідацією Pydantic-схемою перед записом.

Розгортання PostgreSQL у контейнері Docker обрано заради відтворюваності середовища — однакові версія, конфігурація й набір розширень на машині розробника та на стенді захисту — і простоти початкового налаштування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Обґрунтовані в попередньому розділі методи оптимізації та обрані технології втілено в працездатну вебсистему, будову якої розглянуто далі — від серверного ядра до клієнтського застосунку. У цьому розділі описано внутрішню організацію серверної частини та склад REST API (підрозділ 3.1), логічну структуру реляційної бази даних (підрозділ 3.2), програмну реалізацію функціональних модулів системи (підрозділ 3.3) і характеристику поточного стану розробки (підрозділ 3.4).

3.1 Внутрішня організація серверної частини та специфікація REST API

Серверна частина має внутрішню організацію з п’яти шарів: маршрутизація HTTP-запитів; валідація та серіалізація за контрактами; бізнес-логіка (класифікація типу задачі, рекомендація методу, формування PDF-звітів); обчислювальне ядро (солвери з єдиним інтерфейсом, реєстровані в спільному реєстрі); доступ до бази (ORM-моделі та методи-репозиторії). Допоміжні модулі забезпечують ініціалізацію застосунку, конфігурацію оточення та автентифікацію за JWT.

Розподіл відповідальностей між шарами найзручніше простежити на трасі типового запиту. Виклик POST /solve спочатку потрапляє до HTTP-шару, який зчитує тіло у JSON і передає його у Pydantic-схему — на цьому етапі перевіряється тип задачі, наявність обов’язкових полів і структурна узгодженість формулювання. Далі дискримінатор problem_type спрямовує валідовану модель до сервісу класифікації, що обирає метод і повертає його ім’я разом із поясненням. Реєстр солверів витягає за цим іменем відповідний клас-нащадок BaseSolver, який виконує обчислення на NumPy-матрицях і повертає Solution-об’єкт з оптимумом, послідовністю ітерацій і метаданими виконання. Останній шар записує результат у

таблиці problems і solutions, а HTTP-шар серіалізує відповідь у JSON. Жоден з нижніх шарів не звертається до верхніх — це робить обчислювальне ядро повторно використовуваним.

Така шарова організація відповідає класичному поділу обов'язків у промислових REST-сервісах і дає три практичні переваги: бізнес-шар не залежить від протоколу передачі й може бути повторно використаний (наприклад, у пакетному режимі обробки задач з командного рядка); шар доступу до даних інкапсулює всі звернення до СКБД, що дозволяє в майбутньому замінити PostgreSQL на іншу базу без зміни бізнес-логіки; кожен шар тестується ізольовано — солвери прямо, без HTTP-обгортки, а роутери через httpx.TestClient із підставленим тестовим репозиторієм. Загальну компонентну діаграму системи наведено на рисунку 3.1.

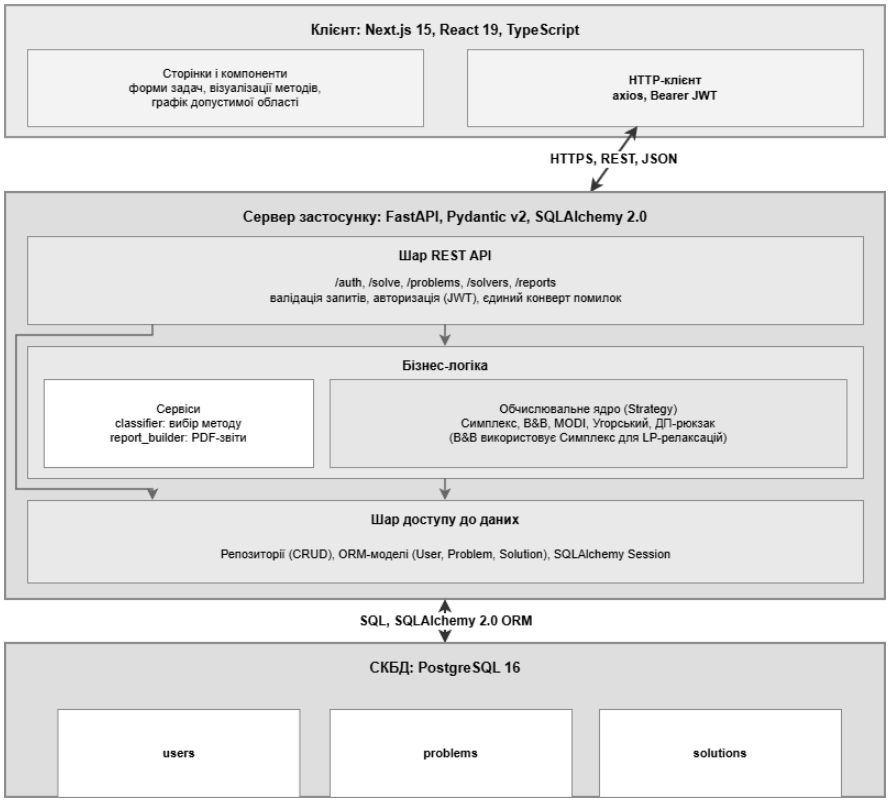

Рисунок 3.1 — Компонентна діаграма системи

Взаємодія клієнта і сервера відбувається за принципами REST: ендпоінти

ідентифікують ресурси через URL, операції відображаються на HTTP-методи, тіла запитів і відповідей передаються у форматі JSON. Усі ендпоінти, крім службових і пов’язаних з реєстрацією, потребують JWT у заголовку Authorization. Помилки повертаються в єдиному форматі: об’єкт error з полями code, message (українською), details. Повний перелік ендпоінтів подано в таблиці 3.1.

Таблиця 3.1 — Перелік ендпоінтів REST API

Метод	Шлях	Призначення
POST	/auth/register	Реєстрація нового користувача
POST	/auth/login	Автентифікація і видача JWT-токена
GET	/auth/me	Профіль поточного користувача
POST	/solve	Розв’язання задачі обраним або рекомендованим методом
GET	/problems	Список задач користувача з фільтрами і пагінацією
GET	/problems/{id}	Повна задача з усіма її розв’язаннями
DELETE	/problems/{id}	Видалення задачі і всіх її розв’язань
GET	/solvers	Каталог зареєстрованих солверів
GET	/reports/problem/{id}	Генерація PDF-звіту за задачею
GET	/health	Перевірка живості процесу

Спільною опорою всіх перелічених ендпоінтів є модель даних, описана в наступному підрозділі.

3.2 Логічна структура реляційної бази даних

Логічна модель побудована навколо трьох сутностей — користувачі, постановки задач і результати розв’язання. Як ідентифікатори обрано тип UUID; позначки часу проставляються базою автоматично. Загальну модель наведено на рисунку 3.2.

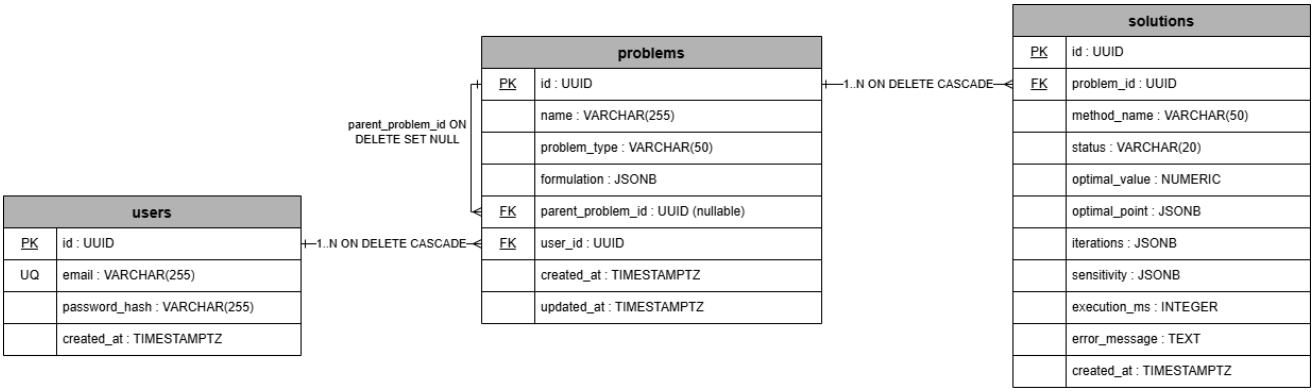

Рисунок 3.2 — Логічна модель бази даних

Логічна модель складається з трьох таблиць: users — для зареєстрованих користувачів системи, problems — для постановок оптимізаційних задач, solutions — для результатів їх розв’язання конкретним методом.

Таблиця users зберігає мінімально необхідні дані для входу: ідентифікатор, електронну адресу як логін, хеш пароля та час реєстрації.

Таблиця problems зберігає сформульовані задачі разом із метаданими — тип задачі, назву, час створення й модифікації, посилання на користувача-власника та на “батьківську” задачу для what-if-варіантів. Поле problem_type містить рядковий дискримінатор типу; список допустимих значень фіксується на рівні застосунку, що дає змогу додавати нові типи задач без міграції. Математичне формулювання — змінні, цільова функція, обмеження або параметри шаблону — зберігається у полі formulation формату JSONB, що дозволяє покривати всі типи задач одним полем.

Таблиця solutions зберігає результат окремого запуску методу: значення цільової функції в оптимумі, координати оптимальної точки, послідовність ітерацій для покрокової візуалізації, результати аналізу чутливості, тривалість виконання та повідомлення про помилку у разі невдалого запуску.

3.3 Програмна реалізація функціональних модулів системи

Обчислювальне ядро реалізовано як набір незалежних солверів, що успадковують спільний клас BaseSolver і реєструються в центральному словнику за іменем методу. Архітектура відповідає патерну Strategy: додавання нового методу не потребує модифікації коду роутерів — достатньо створити клас-нащадок і зареєструвати його.

simplex.py — табличний варіант симплекс-методу з технікою Big-M і правилом Бленда; єдиний солвер, що повертає аналіз чутливості (тіньові ціни, діапазони). branch_and_bound.py — метод гілок і меж зі стратегією best-first; LP-релаксації обчислюються через виклик симплекс-солвера. potentials.py — MODI з автоматичним балансуванням і побудовою опорного плану методом північно-західного кута. hungarian.py — матричний варіант Куна–Мункреса з автоматичним доповненням до квадратної матриці та інверсією для максимізації. dp_knapsack.py — класична двовимірна таблиця DP зі складністю O(n·W); відновлення набору — зворотним проходом. solar_battery.py — доменна обгортка прикладної задачі з енергетики: складання добового графіка зарядки-розрядки акумулятора сонячної електростанції для максимізації прибутку від продажу енергії за змінними тарифами. Задачу сформульовано як ЛП на дискретному часовому горизонті із заданою кількістю слотів (у типовому добовому сценарії — 24 погодинні слоти): змінні відповідають обсягам заряду/розряду в кожному слоті, обмеження — балансам потужності та ємності акумулятора, цільова функція — сумарному прибутку за змінними тарифами [21]. Розв’язання делегується симплекс-солверу, тому solar_battery.py не рахується в переліку п’яти методів, а слугує містком між обчислювальним ядром і прикладною галуззю спеціальності. Симплекс-метод виступає центральним вузлом і використовується трьома способами: як самостійний солвер, як LP-розв’язувач у методі гілок і меж та як бек-енд для модуля сонячної електростанції. Повний програмний код перелічених солверів наведено в додатку А.

Окремий модуль services/classifier.py реалізує сервіс рекомендації методу. Функція recommend_method отримує об’єкт задачі і повертає пару (ім’я методу, пояснення вибору українською). Логіка — дерево рішень: для шаблонних типів метод однозначно визначається класом, для довільного ЛП додатково перевіряється структура змінних на цілочисловість.

3.4 Характеристика поточного стану розробки

Систему реалізовано в обсязі, достатньому для повноцінної демонстрації запропонованого підходу. На сервері повністю реалізовано обчислювальне ядро з п’яти солверів, сервіс рекомендації методу, REST API в повному обсязі, шар роботи з базою (три таблиці, чотири міграції Alembic), цикл автентифікації та модуль генерації PDF-звітів. Клієнтська частина охоплює п’ять груп екранів (автентифікація, інформаційна панель, форми, результати, історія) з окремим типом покрокової візуалізації для кожного методу. Аналіз чутливості реалізовано для лінійних задач, де він має чітку теоретичну основу через двоїстість симплекс-методу. Для цілочислових і комбінаторних задач (гілок і меж, угорський алгоритм, динамічне програмування) поняття чутливості некоректне в класичній постановці, тому для цих методів він не передбачений за дизайном.

Систему перевірено автоматизованими тестами, опис яких наведено в підрозділі 4.3. Емпіричні діапазони розмірності задач, на яких реалізовані методи демонструють стабільну роботу, наведено в таблиці 3.2. Для кожного методу таблиця зіставляє три показники — емпірично перевірену розмірність задачі, встановлений у коді жорсткий ліміт і головний чинник, що стримує подальше зростання розмірності. Ці значення встановлено експериментально — послідовним збільшенням розмірності тестових задач до межі, за якою час відповіді переставав бути прийнятним для інтерактивної роботи.

Таблиця 3.2 — Емпіричні діапазони розмірності задач, на яких реалізовані методи демонструють стабільну роботу

Метод	Перевірена розмірність	Жорсткий ліміт у коді	Обмежувальний фактор
Симплекс-метод	до 20 змінних і 20 обмежень	500 ітерацій	обсяг JSON-даних ітерацій для рендерингу
Метод гілок і меж	до 10 цілочислових змінних, MILP до 15+5	1000 вузлів дерева	експоненційне зростання дерева у гіршому випадку
Метод потенціалів	транспортні матриці до 10×10	200 ітерацій перебудови	складність пошуку циклу
Угорський алгоритм	матриці до 10×10	200 зовнішніх ітерацій	кубічна обчислювальна складність
Динамічне програмування	до 20 предметів, місткість рюкзака до 1000	без явного ліміту	псевдополіноміальна складність O(n·W)

Жорсткі ліміти в коді істотно перевищують перевірені діапазони і обрано з огляду на навчальне призначення системи — вони радше запобіжник проти зависання процесу, ніж реальне обмеження розмірності. Для ітераційних методів — симплекс-методу, методу потенціалів і угорського алгоритму — перевищення ліміту перериває обчислення з поверненням повідомлення про помилку з ясним текстом. Метод гілок і меж при досягненні ліміту вузлів завершує перебір і повертає найкращий знайдений на цей момент допустимий розв’язок; на перевірених навчальних розмірностях задач цей ліміт практично недосяжний.

4 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

У цьому розділі описано практичні аспекти використання розробленої вебсистеми. Викладено мінімальні апаратні й програмні вимоги до серверної та клієнтської частин, послідовність дій з розгортання застосунку у виробничому середовищі, типовий наскрізний сценарій взаємодії користувача з інтерфейсом — від автентифікації до експорту PDF-звіту, описано тестування системи, а також обговорено перспективи подальшого розвитку системи.

4.1 Системні вимоги та інсталяція

Розроблена система складається з трьох взаємодіючих компонентів — серверного застосунку, який реалізує обчислювальне ядро та надає REST-інтерфейс для клієнта; клієнтського застосунку, що виконується у браузері й забезпечує взаємодію з користувачем; системи керування базою даних, у якій зберігаються постановки задач, результати розв’язання та облікові записи користувачів. На етапі апробації всі три компоненти запускаються на одній машині й комунікують через локальні мережеві адреси.

Для роботи з системою кінцевому користувачеві достатньо сучасного браузера та мережевого з’єднання із серверним застосунком. Принципових обмежень на конкретний браузер не передбачено; перевірку працездатності виконано у поширених браузерах настільних операційних систем. Адресу серверного застосунку задано у конфігураційному файлі клієнта, що дозволяє за потреби перенаправити інтерфейс на інший сервер. Серверна частина потребує встановленого середовища виконання мови Python і доступної СКБД PostgreSQL; для типових навчальних задач спеціальних апаратних вимог до сервера не висувається.

4.2 Сценарій роботи користувача з системою

Наскрізний сценарій роботи з системою охоплює п’ять послідовних етапів: автентифікацію, формулювання оптимізаційної задачі, отримання результату розв’язання, ознайомлення з покроковою візуалізацією та аналізом чутливості, повернення до збереженої історії й експорт PDF-звіту. Нижче кожен етап ілюструється відповідними елементами користувацького інтерфейсу.

Першим етапом роботи з системою є автентифікація. Користувач отримує доступ до особистого кабінету через форму входу, що приймає електронну адресу та пароль (рисунок 4.1). За відсутності облікового запису посилання у нижній частині форми переадресовує на сторінку реєстрації, де запитуються ті самі поля з додатковим підтвердженням пароля.

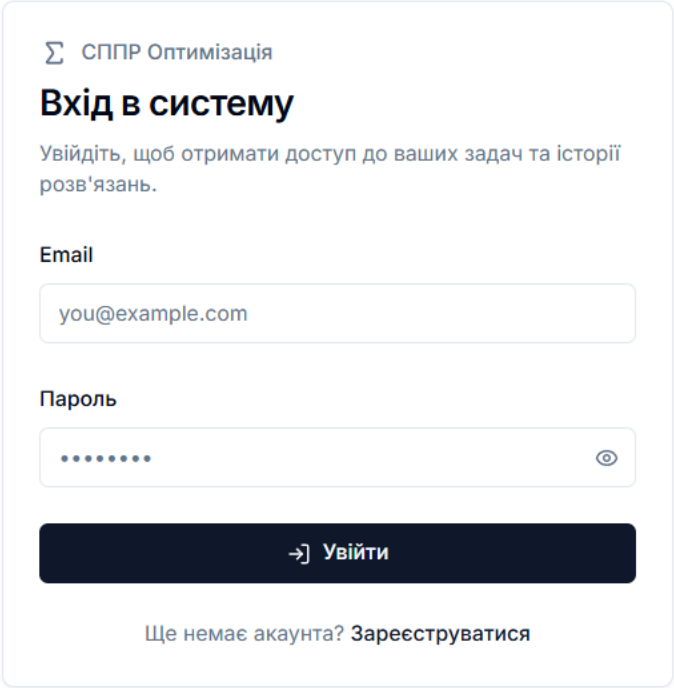

Рисунок 4.1 — Форма входу в систему

Після успішної автентифікації користувач переходить на сторінку формулювання задачі, де доступні два логічні режими (рисунок 4.2). Перший —

“довільна задача”, позначена як рекомендований початок: користувач уводить вектор змінних, цільову функцію і систему обмежень, після чого сервіс класифікації автоматично визначає тип задачі за її структурою й обирає метод — симплекс для неперервних змінних або гілок і меж для цілочислових. Другий — шаблонні задачі з фіксованим методом: транспортна задача, задача про призначення, задача про рюкзак, а також дві прикладні постановки (планування виробництва і добового графіка станції з акумулятором), кожна з яких розв’язується специфічно дібраним алгоритмом без додаткового вибору з боку користувача.

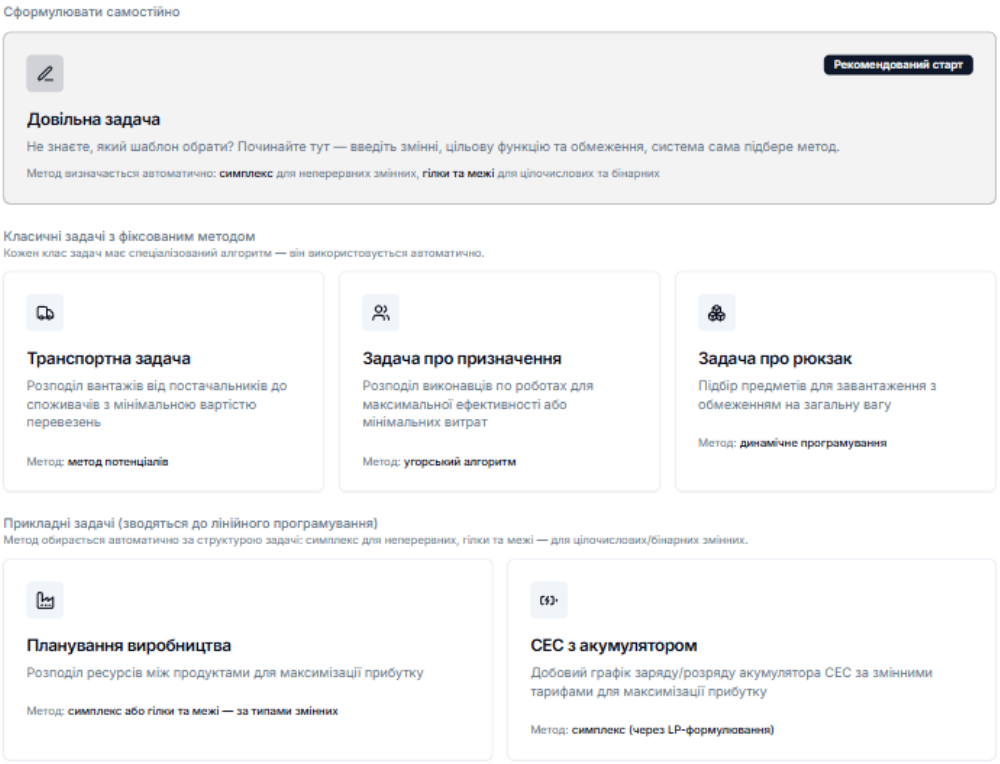

Рисунок 4.2 — Сторінка вибору типу задачі

Після вибору типу задачі система відкриває спеціалізовану форму введення, поля якої відповідають математичній структурі обраного класу. Розгляньмо вигляд цієї форми для кожного з варіантів окремо.

У режимі довільної задачі лінійного програмування форма поділена на три

логічні блоки (рисунок 4.3). У блоці “Змінні” користувач задає перелік змінних задачі: для кожної змінної вказується ім’я, тип (неперервна, цілочислова або бінарна) та межі допустимих значень. У блоці “Цільова функція” обирається напрямок оптимізації — максимізація або мінімізація — і вводяться коефіцієнти при кожній зі змінних; під полями введення відображається попередній перегляд цільового виразу у математичній нотації. У блоці “Обмеження” формуються рядки лінійних нерівностей: для кожного обмеження вказуються коефіцієнти при змінних, знак (≤, =, ≥) і права частина. Кількість змінних і обмежень не фіксована: відповідні кнопки додають нові рядки, а елемент “кошик” вилучає зайві. Метод розв’язання визначається автоматично за наявністю цілочислових або бінарних змінних — симплекс для повністю неперервних задач і гілок і меж в інших випадках.

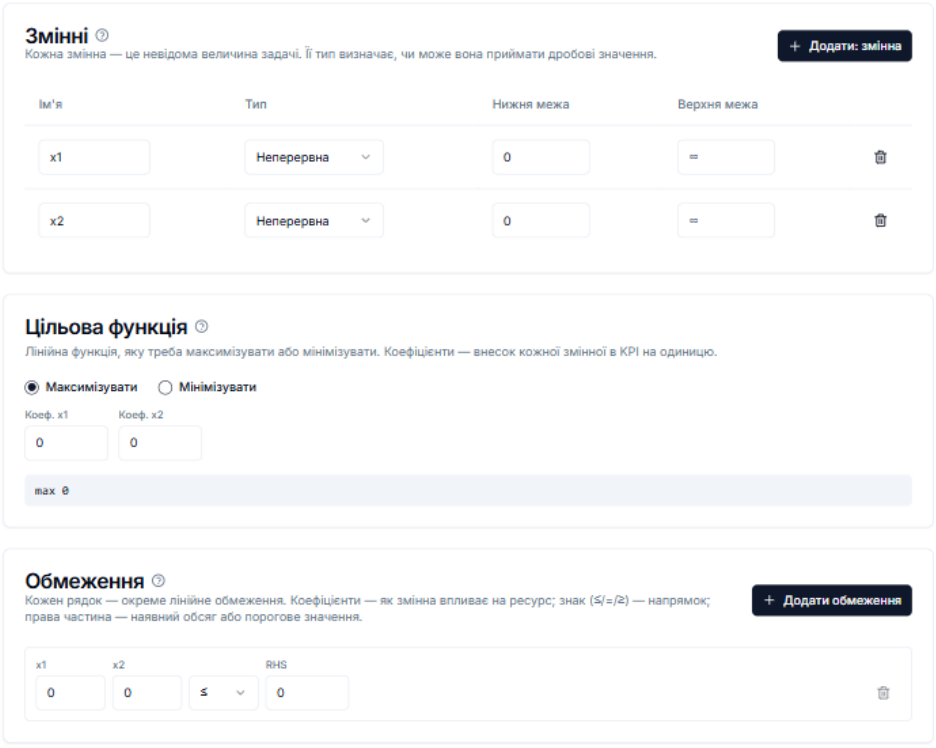

Рисунок 4.3 — Форма довільної задачі лінійного програмування

Форма транспортної задачі повторює математичну специфіку класу (рисунок

4.4). Користувач формує список постачальників із вказівкою обсягу пропозиції кожного, список споживачів із вказівкою попиту, а матриця вартостей перевезень показується як спільна таблиця з іменами постачальників за рядками й споживачами за стовпцями. Над матрицею система виводить індикатор балансу — сумарну пропозицію і сумарний попит з відміткою про збалансованість задачі; у разі дисбалансу обчислювальне ядро автоматично доповнює задачу фіктивним постачальником або споживачем з нульовими тарифами. Метод розв’язання — метод потенціалів — задається жорстко через специфіку класу, без можливості вибору з боку користувача.

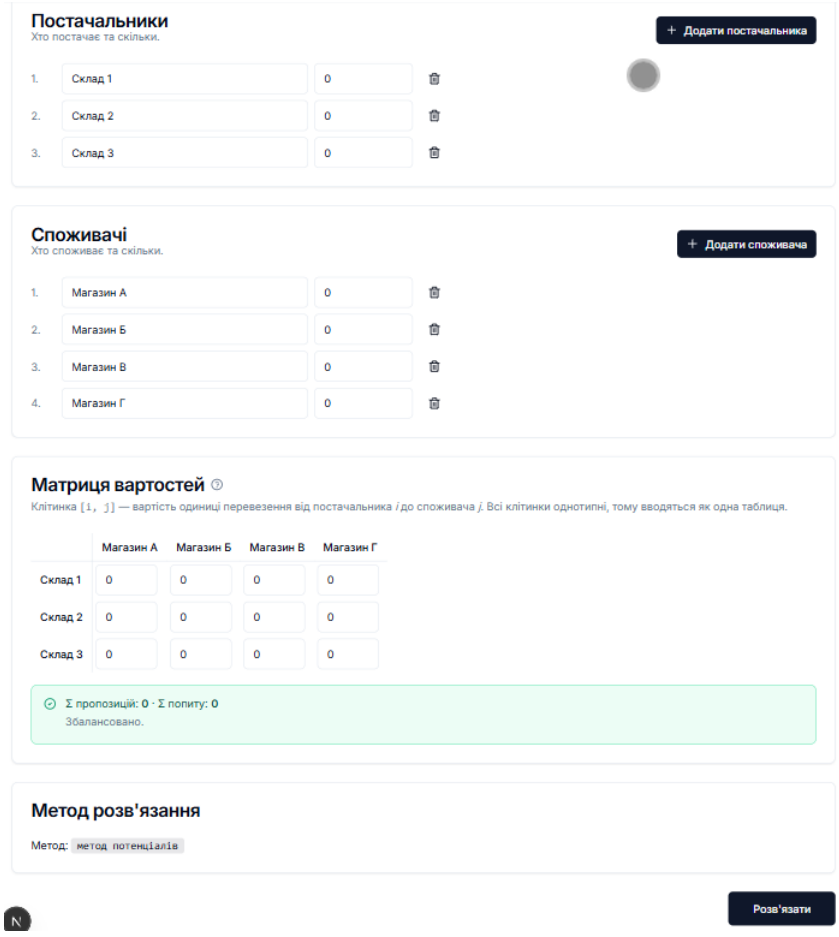

Рисунок 4.4 — Форма транспортної задачі

Форма задачі про призначення побудована за аналогічним принципом

(рисунок 4.5). Користувач формує список виконавців і список завдань, а матриця ефективності або вартостей задається спільною таблицею з виконавцями за рядками й завданнями за стовпцями. Радіокнопки під матрицею задають напрямок оптимізації — мінімізація вартості або максимізація ефективності; у разі максимізації обчислювальне ядро виконує внутрішнє перетворення матриці (інверсію елементів відносно максимуму) перед застосуванням алгоритму. Для прямокутних матриць — коли кількість виконавців не дорівнює кількості завдань — задача автоматично доповнюється фіктивними рядками або стовпцями. Метод розв’язання — угорський алгоритм — задається жорстко.

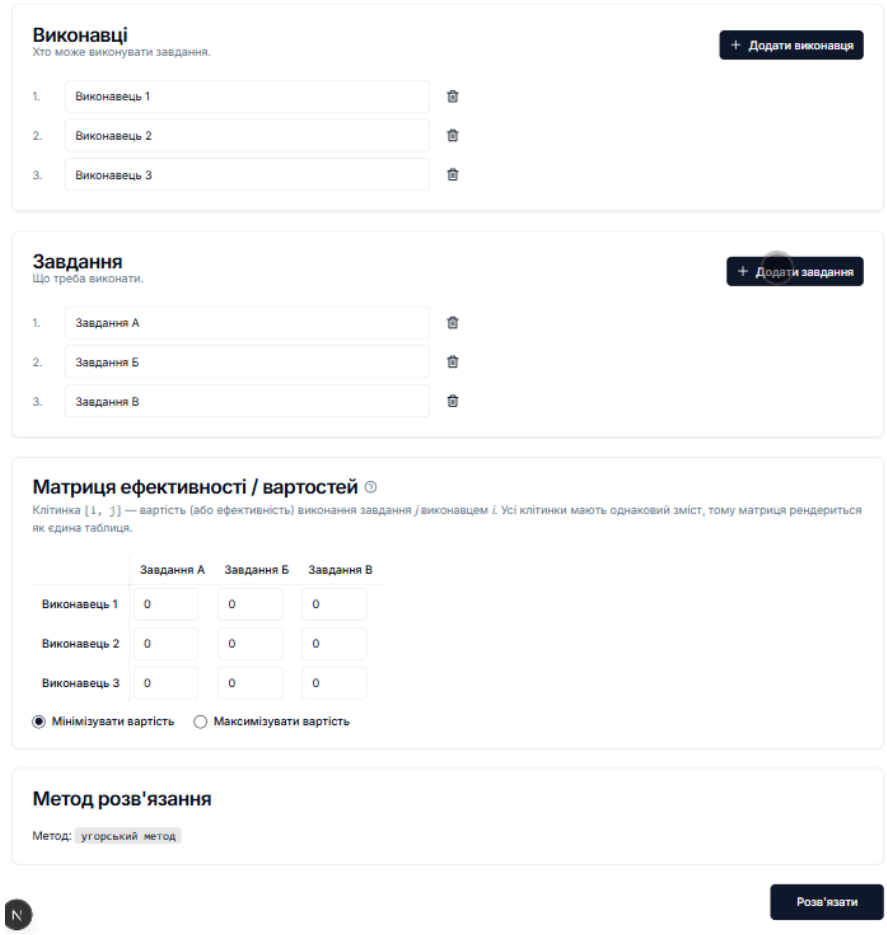

Рисунок 4.5 — Форма задачі про призначення

Форма задачі про рюкзак найкомпактніша з шаблонних (рисунок 4.6).

Користувач задає максимальну сумарну вагу — місткість рюкзака — і перелік предметів, кожен з яких описується іменем, вагою та цінністю; кнопка додавання дозволяє розширити перелік довільною кількістю елементів. Метод розв’язання — динамічне програмування — задається жорстко через класову специфіку задачі. Така форма прозоро відображає математичну постановку задачі про рюкзак у класичному формулюванні 0/1.

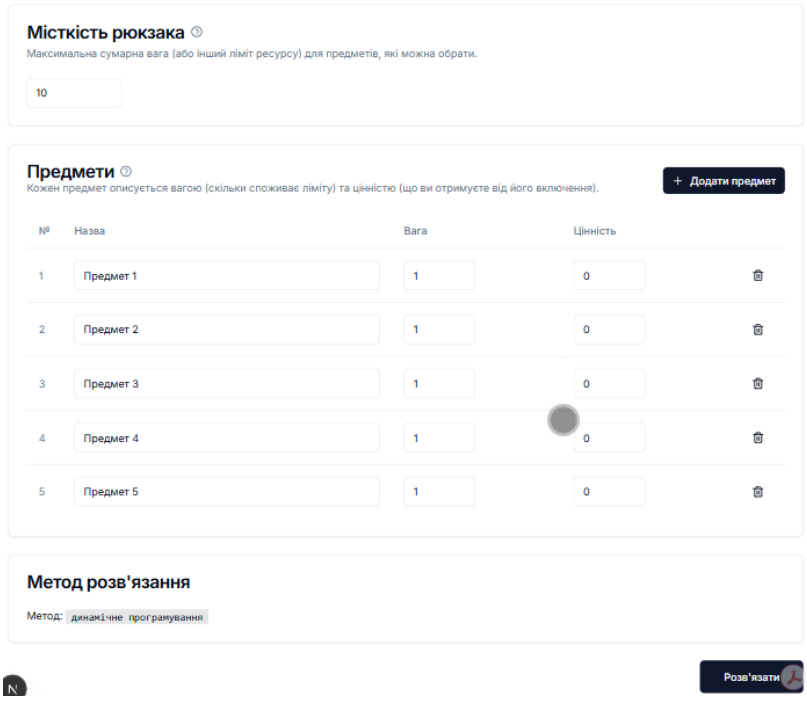

Рисунок 4.6 — Форма задачі про рюкзак

Після натискання кнопки “Розв’язати” система виконує обчислення на стороні сервера і відкриває сторінку результату (рисунок 4.7). У верхній частині сторінки розташовано заголовок задачі з її іменем, тип задачі, відмітку часу створення і три дії над задачею: клонування для what-if-варіантів, експорт PDF-звіту, видалення з історії. Нижче — чотири вкладки змісту: “Рішення”, “Покрокове розв’язання”, “Графічна візуалізація”, “Аналіз чутливості”.

На вкладці “Рішення” подано повну математичну модель задачі у стандартній нотації лінійного програмування — напрямок оптимізації цільової функції, система

обмежень, умови невід'ємності змінних. Далі великим шрифтом виведено оптимальне значення цільової функції зі статусом задачі — "Оптимально", "Несумісно" або "Необмежено" — обраним методом і часом виконання у мілісекундах. Під значенням наведено коротке пояснення вибору методу, наприклад "Усі змінні неперервні, задача лінійна — симплекс-метод". Завершує вкладку блок "Оптимальна точка" — таблиця змінних із значеннями, прийнятими в оптимумі.

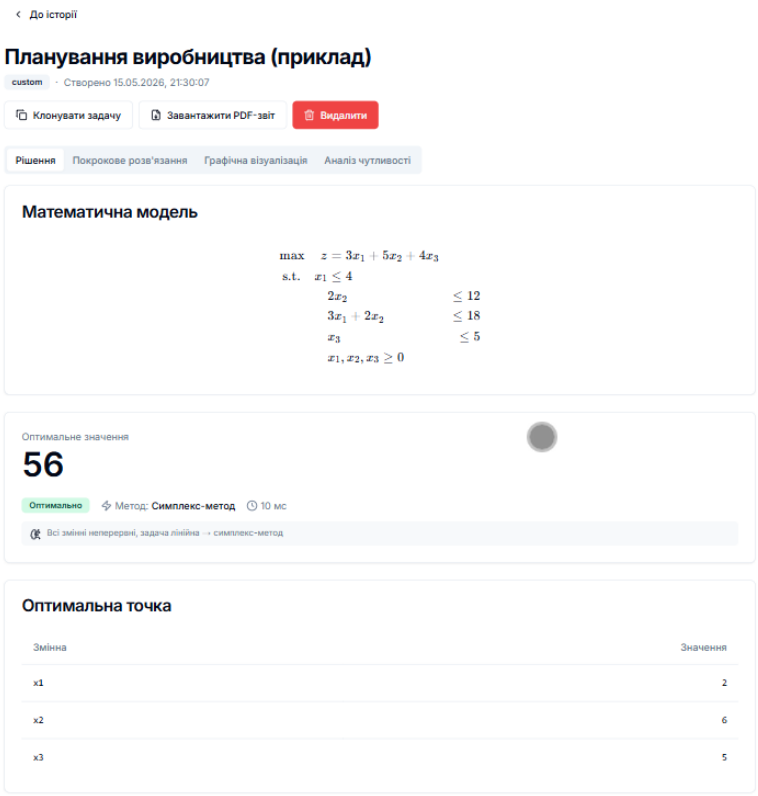

Рисунок 4.7 — Сторінка результату розв'язання задачі

Вкладка "Покрокове розв'язання" відображає послідовність ітерацій обраного методу у формі, специфічній для класу задачі (рисунок 4.8). У верхній частині вкладки розташовано індикатор поточного кроку ("Крок 1 з 5") і регулятор швидкості анімації — 0,5×, 1×, 2×. Безпосередньо під ним наводиться поточний стан обчислювального процесу. Для симплекс-методу це симплекс-таблиця з

рядками базисних змінних і стовпцями всіх — як справжніх, так і допоміжних та штучних — змінних задачі; над таблицею показано опис поточної ітерації, під таблицею — поточне значення цільової функції і поточна точка у просторі змінних. Знизу розташовано панель навігації — кнопки “Попередній”, “Наступний”, “Програти” і кінцеві позиції переходу на початок або в кінець послідовності, доповнені смугою прокрутки для швидкого переміщення.

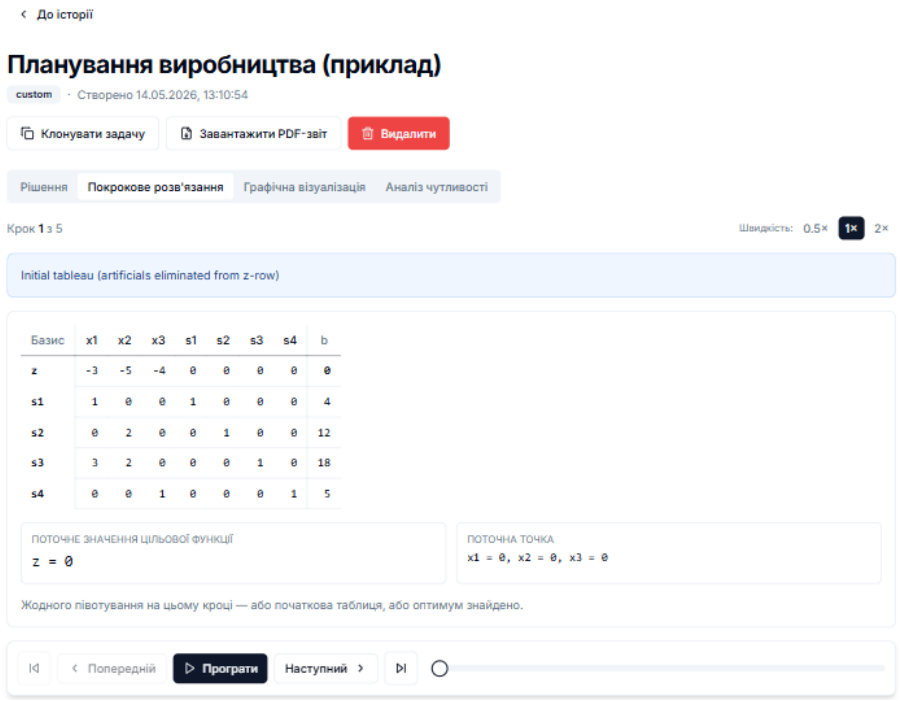

Рисунок 4.8 — Покрокова візуалізація симплекс-методу

Для методу гілок і меж та сама вкладка демонструє побудову дерева вузлів задачі — кожна гілка відповідає введенню додаткового обмеження на цілочислову змінну, а кольори вузлів кодують їхній стан (рисунок 4.9). Зелений вузол позначає кандидата на оптимум (інкумбент), сірий — відсічений за оцінкою або несумісністю, блакитний — поточний активний для розширення. Біля кожного вузла наведено значення LP-релаксації; у нижній частині — лічильники активних вузлів, поточного інкумбенту і кількості відсічених гілок. Подібну логіку має покрокова візуалізація для методу потенціалів, угорського алгоритму та

динамічного програмування — кожен метод відображає себе у тій нотації, що найкраще передає його обчислювальну суть: матриця алокацій із циклом перерозподілу для MODI, матриця з лініями покриття для Hungarian, поетапно заповнювана таблиця для динамічного програмування.

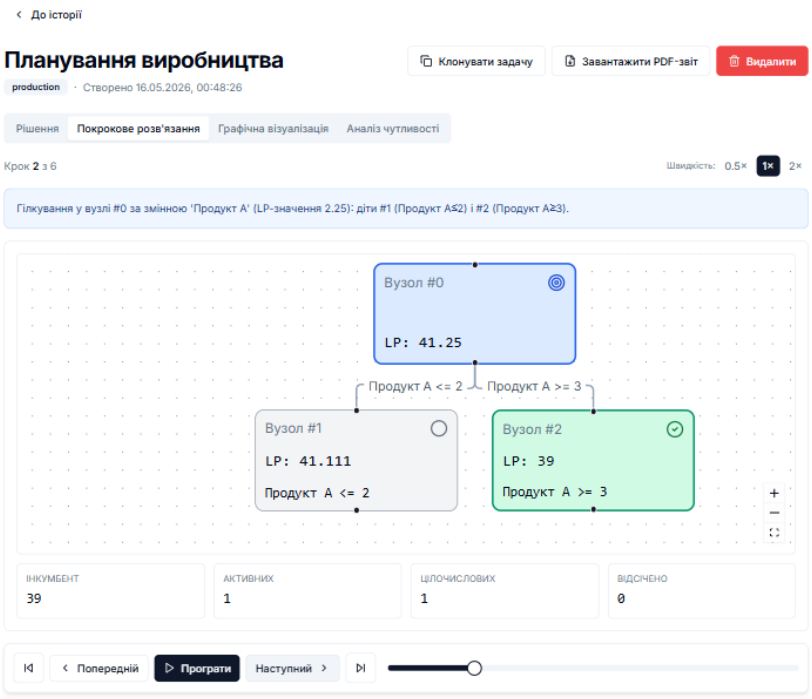

Рисунок 4.9 — Покрокова візуалізація методу гілок і меж

Третя й четверта вкладки сторінки результату — “Графічна візуалізація” та “Аналіз чутливості” — пропонують додаткові способи аналізу одержаного розв’язку.

Вкладка “Графічна візуалізація” розгортає допустиму область задачі лінійного програмування у дво- або тривимірному просторі — залежно від кількості реальних змінних задачі (рисунок 4.10). Прозорим багатогранником зображено допустиму область, синіми ребрами — її межі, сірими точками — вершини, червоним ромбом — знайдений оптимум, зеленою стрілкою — напрямок зростання цільової функції з центра області. Користувач може обертати сцену

мишею для зручного перегляду під різними кутами. Під графіком наведено коротку легенду “Як читати 3D-графік”, що пояснює призначення кожного візуального елемента; для задач лише з двома реальними змінними аналогічну побудову виконано на двовимірній площині.

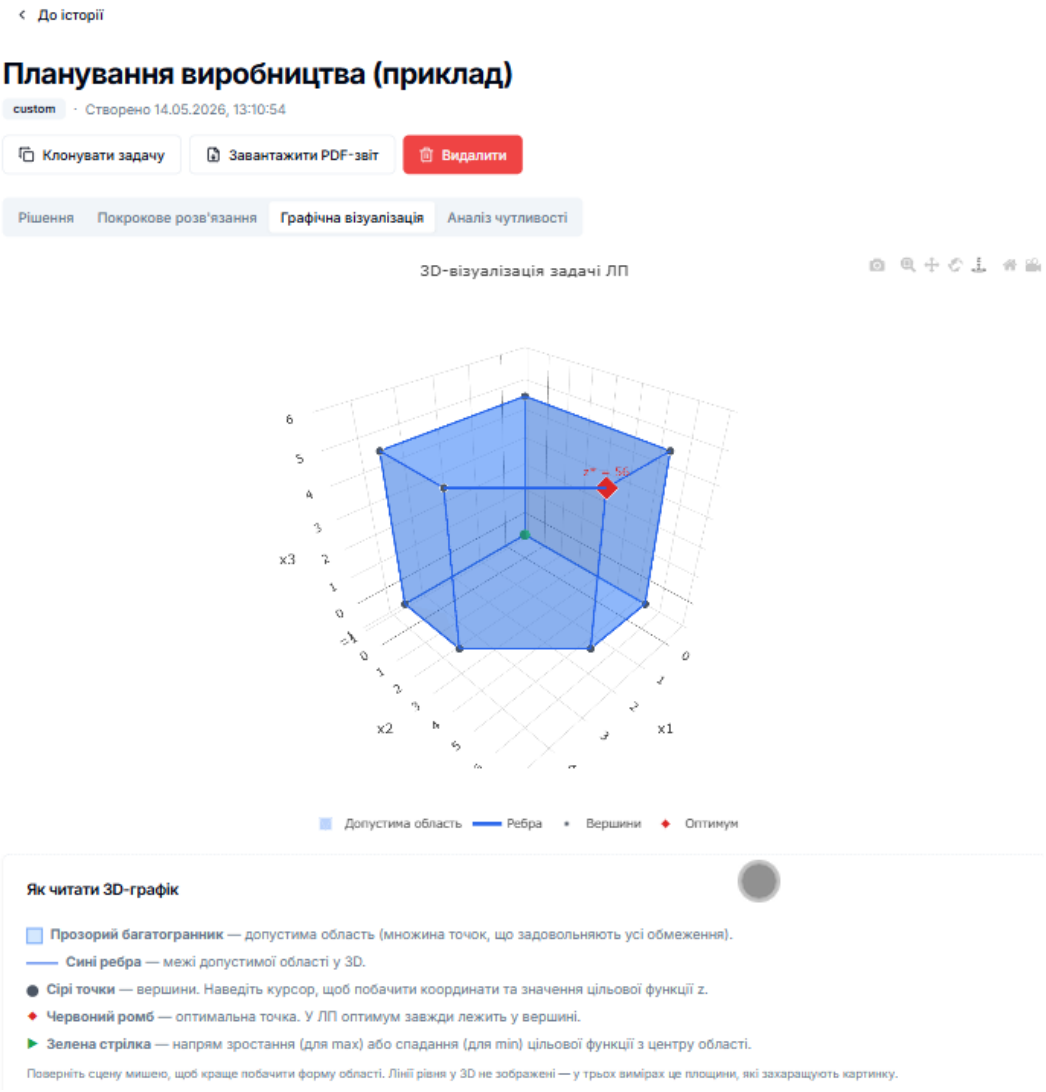

Рисунок 4.10 — Графічна візуалізація допустимої області задачі ЛП

Вкладка “Аналіз чутливості” відображає двоїсту інформацію симплекс-методу — тіньові ціни обмежень і допустимі діапазони зміни правих частин (рисунок 4.11). Горизонтальна штрихова діаграма у верхній частині подає абсолютні значення тіньових цін за всіма обмеженнями задачі, а таблиця нижче —

поточне значення b_i кожного обмеження, тіньову ціну і допустимий діапазон, у межах якого ця ціна залишається сталою. Економічна інтерпретація тіньової ціни — приріст оптимуму при збільшенні правої частини відповідного обмеження на одиницю; нульова тіньова ціна позначає ненасичене обмеження, що за поточної конфігурації не впливає на оптимум.

№	Поточна b_i	Тіньова ціна	Допустимий діапазон b_i
1	4	0	[2; +∞]
2	12	1.5	[6; 18]
3	18	1	[12; 24]
4	5	4	[0; +∞]

Рисунок 4.11 — Аналіз чутливості: тіньові ціни обмежень

Окремою функцією на тій самій вкладці є експериментальна панель “Що буде, якщо…” (рисунок 4.12). Користувач може варіювати правими частинами обмежень і коефіцієнтами цільової функції через слайдери з відображенням оригінального значення, поточного значення і допустимого діапазону. Поки нові значення лежать у межах допустимого діапазону, новий прогнозований оптимум обчислюється миттєво на стороні клієнта через тіньові ціни — без повторного звернення до сервера. Для виходу за межі діапазону передбачено кнопку “Перерозв’язати з новими параметрами”, яка надсилає клоноване формулювання на сервер і повертає точний розв’язок. Така архітектура дозволяє користувачеві швидко досліджувати чутливість оптимуму до варіацій параметрів без повторних

повноцінних обчислень.

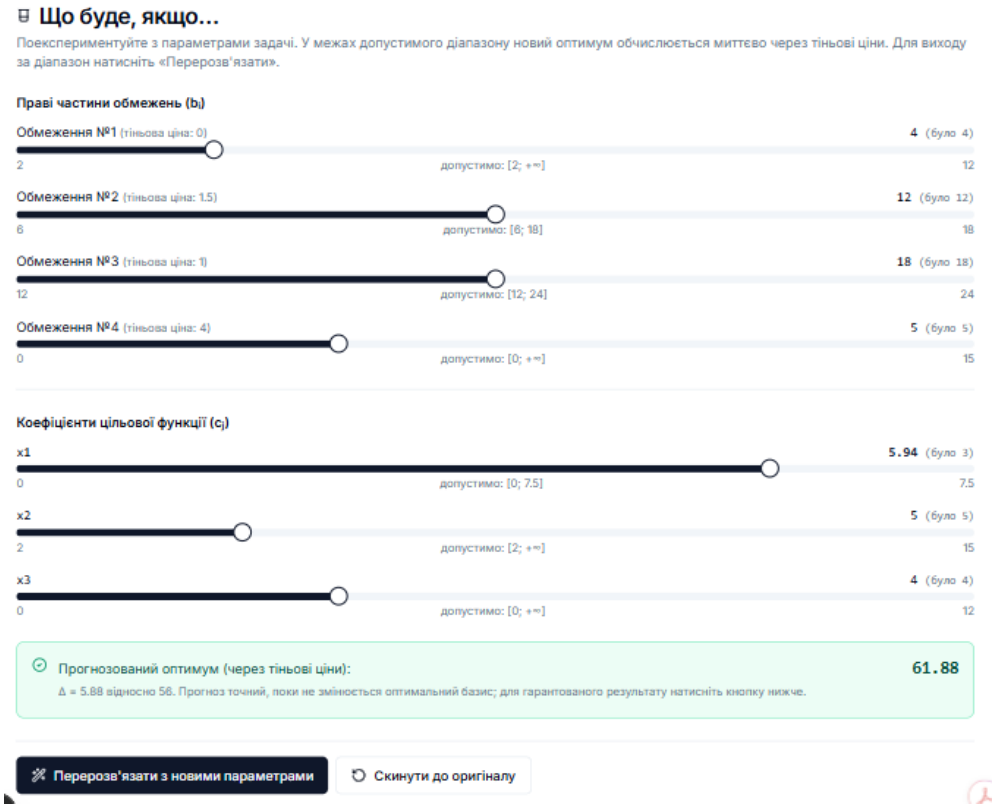

Рисунок 4.12 — Панель what-if-аналізу

Усі сформульовані й розв’язані задачі автоматично зберігаються в персональній історії, прив’язаній до облікового запису користувача (рисунок 4.13). Сторінка історії містить пошук за назвою, фільтри за типом задачі і статусом розв’язку, сортування за датою створення. Для кожної задачі у таблиці виведено її назву, тип, дату створення і статус результату; у стовпці дій — кнопки відкриття повного результату, клонування задачі у новий запис (для what-if-сценаріїв з модифікованими параметрами) і видалення з історії. Видалення задачі каскадно видаляє всі прив’язані до неї розв’язання.

Особливе значення в історії має операція клонування. Натискання відповідної кнопки створює новий запис, у якому збережено посилання на оригінальну задачу через зовнішній ключ parent_problem_id у таблиці problems —

це дозволяє системі утворювати дерева споріднених задач і простежувати, які варіанти походять від яких. Така структура зручна для what-if-сценаріїв: користувач формулює базову задачу, потім створює кілька клонів з різними значеннями параметрів, після чого має історію всіх досліджених варіантів зі збереженням зв’язку з вихідною постановкою.

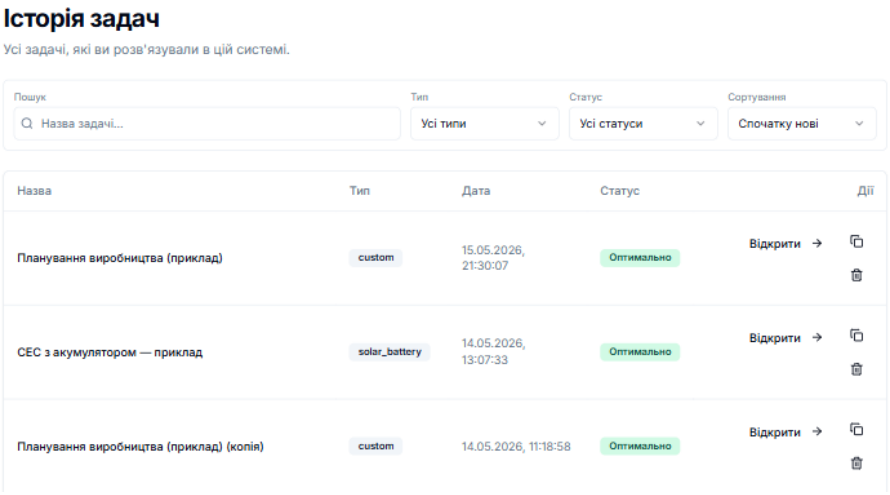

Рисунок 4.13 — Сторінка історії задач

Для офлайн-передачі результатів передбачено експорт PDF-звіту, що формується кнопкою “Завантажити PDF-звіт” на сторінці результату (рисунок 4.14). Звіт містить заголовок з ім’ям задачі та її унікальним ідентифікатором, дату створення і використаний метод, повну постановку задачі — змінні з типами й межами, цільову функцію, систему обмежень — а також основний результат розв’язання з оптимальним значенням, оптимальною точкою і часом виконання. Звіт формується на стороні сервера й одразу завантажується на робочу станцію користувача.

PDF-звіт побудовано як послідовність типових блоків: титульний блок з назвою задачі та її ідентифікатором, розділ постановки задачі з повною математичною моделлю, розділ результату з оптимальним значенням і оптимальною точкою. Такий формат залишається придатним для друку й архівного

зберігання незалежно від наявності у одержувача доступу до самої системи; це підвищує цінність результату для тих сценаріїв, у яких розв'язок задачі має документально супроводжуватися в окремому файлі.

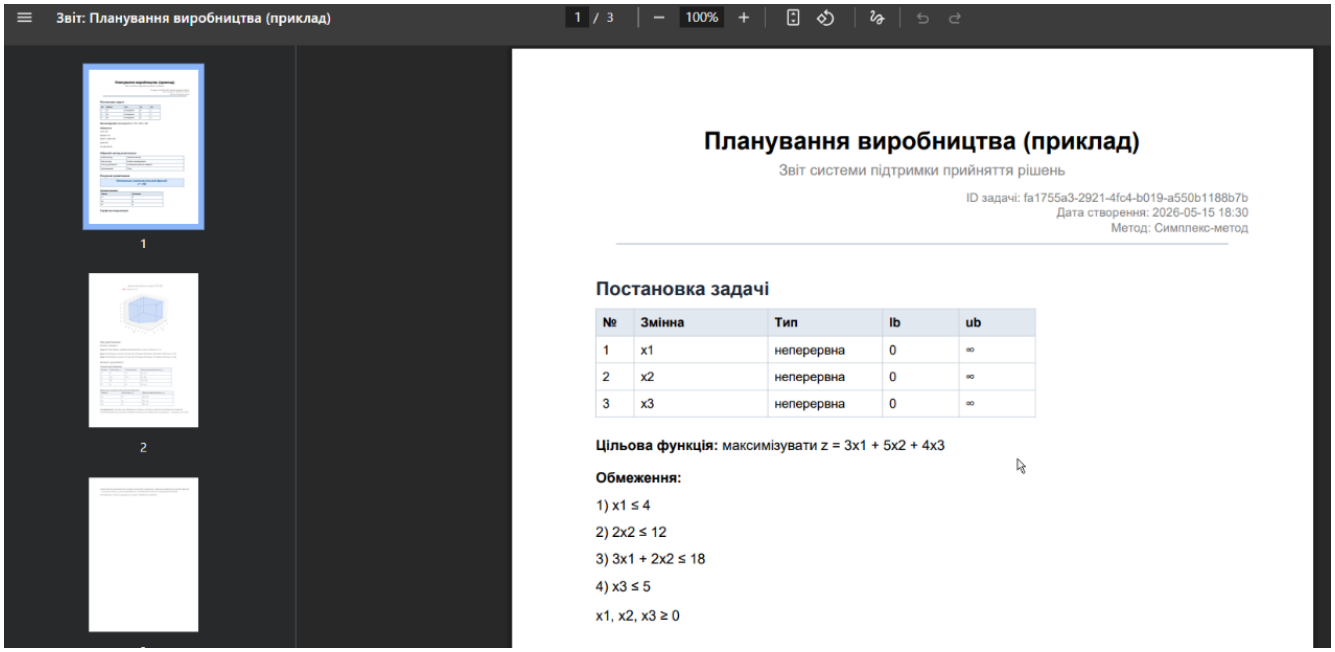

Рисунок 4.14 — Приклад згенерованого PDF-звіту

Описаний наскрізний сценарій — від автентифікації, формулювання задачі та отримання результату через покрокову візуалізацію й аналіз чутливості до повернення до збереженої історії й експорту PDF-звіту — формує цілісне уявлення про практичну роботу з системою. Кожен з п'яти етапів спирається на типи задач і архітектурні рішення, описані в попередніх розділах, і відображає те, як абстрактні математичні постановки з розділу 2 та компоненти програмної реалізації з розділу 3 виявляють себе на рівні безпосередньої взаємодії користувача із системою. Хоча сценарій проілюстровано на конкретних прикладах, його послідовність однакова для всіх п'яти класів задач: змінюється лише форма введення на другому етапі, тоді як отримання результату, покрокова візуалізація, аналіз і збереження в історії відбуваються в єдиний спосіб.

4.3 Тестування системи

Оскільки обчислювальне ядро реалізовано власноруч, а не на готових бібліотеках оптимізації, на систему лягає обов'язок довести коректність цих реалізацій: разом із готовими солверами було відкинуто й перевірені гарантії правильності, які вони дають. Тому в основу тестування покладено крос-верифікацію — кожен власний солвер розв'язує ту саму задачу, що й перевірений зовнішній еталон, після чого значення цільових функцій звіряються в межах числової похибки. Симплекс-метод і метод потенціалів зіставляються з функцією linprog бібліотеки SciPy (транспортну модель попередньо зведено до лінійної), метод гілок і меж — з функцією milp, угорський алгоритм — з функцією linear_sum_assignment, а динамічне програмування — з повним перебором допустимих комбінацій. Перевіряється не наперед задана відповідь, а збіг із незалежно отриманим оптимумом на широкому наборі задач — саме це засвідчує правильність ручних реалізацій.

Окрему групу тестів присвячено не самому розв'язку, а структурі проміжних ітерацій, бо власні солвери писалися насамперед заради покрокової візуалізації. Ці тести стежать, щоб симплекс-таблиці, дерево вузлів, потенціали з циклами перерозподілу, лінії покриття та таблиця динамічного програмування зберігалися у сталому, задокументованому форматі й не руйнувалися при подальших змінах коду.

Решта перевірок захищає межу системи — контракти REST API: що тип задачі правильно розпізнається за полем problem_type, що некоректні вхідні дані (коефіцієнти-нескінченності, порожні масиви, неузгоджені розмірності) відхиляються з типізованою помилкою ще до обчислень, а необмежені й несумісні задачі виявляються коректно. Сюди ж належать тести автентифікації та формування PDF-звіту. Серверний набір реалізовано фреймворком pytest, і він налічує 109 перевірок в одинадцяти файлах, що покривають обчислювальне ядро, REST API, цикл автентифікації та формування звітів.

4.4 Перспективи розвитку системи

Попри повну функціональність у заявлених межах, система має кілька природних напрямків подальшого розвитку — частина з яких виявлена під час реалізації як обмеження поточної архітектури, частина — як можливості розширення прикладного потенціалу системи.

Першочерговим напрямком є підготовка системи до виробничого розгортання. Поточна архітектура передбачає локальний запуск на машині розробника й використання тестових облікових даних бази даних; для практичного застосування на підприємстві необхідно винести конфігурацію секретів у захищене сховище, додати шар оберненого проксі-сервера з підтримкою HTTPS (наприклад, nginx або Traefik), налаштувати періодичне резервне копіювання бази даних і моніторинг доступності сервісу. Архітектурно код до цього готовий — налаштування підключення до бази та параметри JWT уже винесено у файл оточення.

Другим напрямком є розширення набору підтримуваних класів задач. Поточні п’ять методів охоплюють найпоширеніші постановки лінійної й комбінаторної оптимізації, проте поза їх межами залишаються задача про розкрій, багатоцільова оптимізація з компромісними фронтами Парето, потокові задачі (maximum-flow, minimum-cost flow) і задачі мережевого планування PERT/CPM. Архітектура обчислювального ядра дозволяє додавати нові методи без модифікації існуючого коду — достатньо реалізувати клас-нащадок спільного інтерфейсу солверів і описати схему валідації нової постановки задачі.

Третій напрямок — оптимізація обчислювального ядра для задач середньої та великої розмірності. Поточна реалізація зберігає в полі iterations таблиці solutions повний знімок симплекс-таблиці або стану дерева гілок і меж на кожному кроці; для задач з кількома тисячами ітерацій обсяг JSON-документа стає основним обмежувальним фактором. Перспективним напрямком є реалізація ледачої серіалізації — збереження ітерацій лише на запит покрокової візуалізації, з

кешуванням у проміжному сховищі — і паралельного обчислення вузлів дерева гілок і меж з використанням пулу процесів.

Четвертий напрямок стосується розширення аналізу чутливості за межі лінійних задач. Класична теорія чутливості, реалізована в системі для симплекс-методу через двоїстість, не має прямого узагальнення на цілочислове програмування й динамічне програмування — а саме на ці задачі найчастіше припадає прикладне навантаження у логістичних і складських сценаріях. Можливим розв’язанням є реалізація апроксимаційного аналізу через автоматичне перерозв’язання задачі з варіюванням параметрів у заданих діапазонах і агрегацію результатів у тіньовий поверхневий граф.

П’ятий напрямок — інтеграційні можливості із зовнішніми системами. Поточна реалізація передбачає лише ручне введення задачі через веб-форму; на практиці більшість вхідних даних уже існує в електронних таблицях, ERP-системах або базах даних обліку. Перспективним є додавання імпорту постановки задачі з Excel-файлу за фіксованим шаблоном, експорту результату у формат LaTeX (для включення у звіти й публікації) та надання сервісного REST-інтерфейсу для машинної інтеграції із зовнішніми системами без участі веб-інтерфейсу.

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено веборієнтовану систему підтримки прийняття рішень на основі класичних методів оптимізації. Отримано такі основні результати.

1. Проаналізовано прикладні задачі оптимального розподілу обмежених ресурсів і наявні програмні засоби їх розв’язання — промислові алгебраїчні мови та обчислювальні солвери, відкриті оптимізаційні бібліотеки, вбудовані засоби електронних таблиць і вузькоспеціалізовані веб-калькулятори. Установлено, що жоден із наявних класів інструментів не поєднує одночасно широкого охоплення задач різних класів, прозорості проміжних обчислень і доступності без встановлення програмного забезпечення, що визначило неперекриту нішу для розроблюваної системи.

2. Розглянуто математичні постановки та методи розв’язання п’яти класів задач математичного програмування — лінійного та цілочислового програмування, транспортної задачі, задачі про призначення та задачі про рюкзак. Для кожного класу обґрунтовано вибір методу, придатного до покрокової візуалізації: симплекс-методу, методу гілок і меж, методу потенціалів, угорського алгоритму та динамічного програмування.

3. Спроєктовано архітектуру вебсистеми як сукупність серверної частини з обчислювальним ядром і REST-інтерфейсом, клієнтської частини, що виконується у браузері, та реляційного сховища даних. Визначено специфікацію REST API і логічну структуру бази даних для зберігання постановок задач, результатів розв’язання та облікових записів користувачів.

4. Реалізовано обчислювальне ядро з п’яти самостійно розроблених солверів, які фіксують проміжні ітерації обчислень. Це забезпечило покрокову візуалізацію процесу розв’язання, специфічну для кожного методу, — симплекс-таблиці, дерево гілкування, послідовні плани перевезень і матриці призначень.

5. Реалізовано сервіс автоматичної класифікації типу задачі та рекомендації

методу розв’язання, який за формалізованою постановкою визначає відповідний клас задачі й добирає метод, супроводжуючи вибір текстовим обґрунтуванням українською мовою.

6. Розроблено клієнтський інтерфейс, що охоплює формулювання задачі, покрокову візуалізацію обчислень, графічну побудову допустимої області, аналіз чутливості для лінійних задач і ведення персональної історії розв’язань із можливістю повторного аналізу.

7. Реалізовано механізм автентифікації користувачів з ізоляцією персональних даних, збереження сформульованих задач та їх розв’язків, а також генерацію звітів у форматі PDF.

8. Виконано автоматизоване тестування системи в обсязі 109 тест-кейсів, що покривають обчислювальні модулі, REST API, цикл автентифікації та формування звітів. Коректність кожного з п’яти реалізованих методів підтверджено крос-верифікацією проти еталонних реалізацій бібліотеки SciPy.

Перспективи подальшого розвитку системи пов’язані з розширенням набору підтримуваних класів задач, зокрема задачі розкрою методом генерації стовпців, інтеграцією методів багатоцільової оптимізації для пошуку Парето-оптимальних розв’язань і створенням модуля порівняльного аналізу методів за розмірністю задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AMPL : A Mathematical Programming Language. URL: https://ampl.com/ (дата звернення: 15.05.2026).
2. GAMS : General Algebraic Modeling System. URL: https://www.gams.com/ (дата звернення: 15.05.2026).
3. AIMMS : Prescriptive Analytics Platform. URL: https://www.aimms.com/ (дата звернення: 15.05.2026).
4. Gurobi Optimizer. URL: https://www.gurobi.com/ (дата звернення: 15.05.2026).
5. IBM ILOG CPLEX Optimization Studio. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio (дата звернення: 15.05.2026).
6. Pyomo : Python Optimization Modeling Objects. URL: https://www.pyomo.org/ (дата звернення: 15.05.2026).
7. PuLP : a Python Linear Programming API. URL: https://coin-or.github.io/pulp/ (дата звернення: 15.05.2026).
8. JuMP : Modeling Language for Mathematical Optimization. URL: https://jump.dev/ (дата звернення: 15.05.2026).
9. HiGHS : High Performance Optimization Software. URL: https://highs.dev/ (дата звернення: 15.05.2026).
10. Excel Solver : Frontline Systems. URL: https://www.solver.com/ (дата звернення: 15.05.2026).
11. PHPSimplex : Online Tool for Solving Linear Programming Problems. URL: https://www.phpsimplex.com/en/ (дата звернення: 15.05.2026).
12. Wolfram Demonstrations Project. URL: https://demonstrations.wolfram.com/ (дата звернення: 15.05.2026).
13. Vanderbei R. J. Linear Programming : Foundations and Extensions. 5th ed. Cham : Springer, 2020. 471 p.

14. Dantzig G. B. Linear Programming and Extensions. Princeton : Princeton University Press, 1963. 627 p.

15. Hillier F. S., Lieberman G. J. Introduction to Operations Research. 11th ed. New York : McGraw-Hill Education, 2021. 1048 p.

16. Bland R. G. New finite pivoting rules for the simplex method. Mathematics of Operations Research. 1977. Vol. 2, № 2. P. 103–107.

17. Land A. H., Doig A. G. An automatic method of solving discrete programming problems. Econometrica. 1960. Vol. 28, № 3. P. 497–520.

18. Сікора Я. Б., Щехорський А. Й., Якимчук Б. Л. Методи оптимізації та дослідження операцій : навч. посіб. Житомир : ЖДУ ім. І. Франка, 2019. 164 с.

19. Kuhn H. W. The Hungarian method for the assignment problem. Naval Research Logistics Quarterly. 1955. Vol. 2, № 1–2. P. 83–97.

20. Bellman R. Dynamic Programming. Princeton : Princeton University Press, 1957. 342 p.

21. Bean R., Khan H. Using solar and load predictions in battery scheduling at the residential level. 2018. 6 p. URL: https://arxiv.org/abs/1810.11178 (дата звернення: 15.05.2026).

22. Скакодуб С. О. Вебсистема підтримки прийняття рішень на основі методів оптимізації. Modernization of today’s science: experience and trends : матеріали X Міжнар. наук.-теорет. конф. (м. Глазго, 22 трав. 2026 р.). Глазго : SCIENTIA, 2026. С. 155–157.

ДОДАТОК А

“Вебсистема підтримки прийняття рішень на основі методів оптимізації”

УКР.НТУУ “КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР_23

Текст програми

Мова програмування – Python

Аркушів 5

Київ – 2026

Програмні засоби:

– мова програмування – Python;

– операції з матрицями та векторами – бібліотека NumPy;

– черга з пріоритетом у методі гілок і меж – модуль heapq стандартної бібліотеки;

– математичні функції – модуль math стандартної бібліотеки.

simplex.py

```
import numpy as np


EPS, M_BIG, MAX_ITER = 1e-9, 1.0e6, 500



def simplex(c, A, b, senses, sense="maximize"):
    c = -c.copy() if sense == "minimize" else c.copy()
    m, n = A.shape
    A, b, senses = A.copy(), b.copy(), list(senses)
    flipped = [False] * m
    for i in range(m):
        if b[i] < 0:
            A[i], b[i] = -A[i], -b[i]
            if senses[i] == "<=":
                senses[i] = ">="
            elif senses[i] == ">=":
                senses[i] = "<="
            flipped[i] = True
    artif = [i for i in range(m) if senses[i] in ("=", ">=")]
    n_total = n + m + len(artif)
    A_ext = np.zeros((m, n_total))
    A_ext[:, :n] = A
    for i in range(m):
        if senses[i] == "<=":
            A_ext[i, n + i] = 1.0
        elif senses[i] == ">=":
            A_ext[i, n + i] = -1.0
    for k, i in enumerate(artif):
        A_ext[i, n + m + k] = 1.0
    basis = [n + i if senses[i] == "<=" else n + m + artif.index(i)
             for i in range(m)]
    c_full = np.zeros(n_total)
    c_full[:n] = c
    for k in range(len(artif)):
        c_full[n + m + k] = -M_BIG
    T = np.zeros((m + 1, n_total + 1))
    T[1:, :-1], T[1:, -1], T[0, :-1] = A_ext, b, -c_full
    for i in range(m):
        if abs(T[0, basis[i]]) > EPS:
            T[0, :] -= T[0, basis[i]] * T[i + 1, :]
    for _ in range(MAX_ITER):
        j_in = next((j for j in range(n_total) if T[0, j] < -EPS), -1)
        if j_in == -1:
            break
        col = T[1:, j_in]
        min_ratio, i_out, i_out_b = float("inf"), -1, -1
        for i in range(m):
            if col[i] > EPS:
                ratio = T[i + 1, -1] / col[i]
                if (ratio < min_ratio - EPS or
                        (ratio < min_ratio + EPS and basis[i] < i_out_b)):
                    min_ratio, i_out, i_out_b = ratio, i, basis[i]
```

```
        if i_out == -1:
            return "unbounded", None, None
        T[i_out + 1, :] /= T[i_out + 1, j_in]
        for r in range(m + 1):
            if r != i_out + 1 and abs(T[r, j_in]) > EPS:
                T[r, :] -= T[r, j_in] * T[i_out + 1, :]
        basis[i_out] = j_in
    x = np.zeros(n)
    for i, bv in enumerate(basis):
        if bv < n:
            x[bv] = T[i + 1, -1]
    z = T[0, -1]
    return "optimal", (-z if sense == "minimize" else z), x
```

branch_and_bound.py

```
import heapq
import math


EPS, MAX_NODES = 1e-6, 1000



def branch_and_bound(solve_lp, integer_idx, is_max):
    def most_fractional(sol):
        best_idx, best = None, 0.5
        for i in integer_idx:
            frac = sol[i] - math.floor(sol[i])
            if EPS < frac < 1 - EPS and abs(frac - 0.5) < best:
                best, best_idx = abs(frac - 0.5), i
        return best_idx


    status, value, sol = solve_lp([])
    if status != "optimal":
        return status, None, None
    inc_val, inc_sol = None, None
    queue, counter = [], 0
    heapq.heappush(queue, (-value if is_max else value, counter, [], sol, value))
    while queue and counter < MAX_NODES:
        _, _, extra, sol, value = heapq.heappop(queue)
        if inc_val is not None and ((is_max and value <= inc_val + EPS) or
                                    (not is_max and value >= inc_val - EPS)):
            continue
        idx = most_fractional(sol)
        if idx is None:
            if (inc_val is None or (is_max and value > inc_val) or
                    (not is_max and value < inc_val)):
                inc_val, inc_sol = value, sol
            continue
        for sgn, rhs in (("<=", math.floor(sol[idx])),
                         (">=", math.ceil(sol[idx]))):
            child = extra + [(idx, sgn, float(rhs))]
            st, val, s = solve_lp(child)
            if st == "optimal":
                counter += 1
                heapq.heappush(queue,
                               (-val if is_max else val, counter, child, s, val))
    return ("optimal" if inc_sol is not None else "infeasible"), inc_val, inc_sol
```

potentials.py

```
import numpy as np


EPS, MAX_ITER = 1e-9, 200



def northwest_corner(supplies, demands, m, n):
```

```
    plan = np.zeros((m, n))
    basis = np.zeros((m, n), bool)
    s, d = list(supplies), list(demands)
    i = j = 0
    while i < m and j < n:
        alloc = min(s[i], d[j])
        plan[i, j], basis[i, j] = alloc, True
        s[i] -= alloc
        d[j] -= alloc
        if abs(s[i]) < EPS and abs(d[j]) < EPS and i + 1 < m and j + 1 < n:
            i += 1
            basis[i, j] = True
            j += 1
        elif abs(s[i]) < EPS:
            i += 1
        else:
            j += 1
    return plan, basis



def compute_potentials(basis, costs, m, n):
    u, v = [None] * m, [None] * n
    u[0] = 0.0
    changed = True
    while changed:
        changed = False
        for i in range(m):
            for j in range(n):
                if not basis[i, j]:
                    continue
                if u[i] is not None and v[j] is None:
                    v[j] = costs[i, j] - u[i]
                    changed = True
                elif v[j] is not None and u[i] is None:
                    u[i] = costs[i, j] - v[j]
                    changed = True
    return u, v



def find_cycle(basis, entering, m, n):
    cells = [(i, j) for i in range(m) for j in range(n) if basis[i, j]]
    cells.append(entering)
    rows, cols = {}, {}
    for c in cells:
        rows.setdefault(c[0], []).append(c)
        cols.setdefault(c[1], []).append(c)
    seen = {entering}


    def dfs(path, move_row):
        last = path[-1]
        for cand in (rows[last[0]] if move_row else cols[last[1]]):
            if cand == last:
                continue
            if cand == entering and len(path) >= 4:
                return path + [entering]
            if cand in seen:
                continue
            path.append(cand)
            seen.add(cand)
            res = dfs(path, not move_row)
            if res is not None:
                return res
            path.pop()
            seen.discard(cand)
        return None


    return dfs([entering], True) or dfs([entering], False)


def potentials(supplies, demands, costs):
    costs = np.array(costs, float)
```

```
    m, n = len(supplies), len(demands)
    plan, basis = northwest_corner(supplies, demands, m, n)
    for _ in range(MAX_ITER):
        u, v = compute_potentials(basis, costs, m, n)
        entering, best = None, -EPS
        for i in range(m):
            for j in range(n):
                if not basis[i, j] and u[i] is not None and v[j] is not None:
                    delta = costs[i, j] - u[i] - v[j]
                    if delta < best:
                        best, entering = delta, (i, j)
        if entering is None:
            break
        cycle = find_cycle(basis, entering, m, n)
        signs = ["+" if k % 2 == 0 else "-" for k in range(len(cycle))]
        minus = [cycle[k] for k in range(1, len(cycle) - 1) if signs[k] == "-"]
        theta = min(plan[c] for c in minus)
        for k in range(len(cycle) - 1):
            plan[cycle[k]] += theta if signs[k] == "+" else -theta
        basis[entering] = True
        for c in minus:
            if plan[c] < EPS:
                basis[c] = False
                break
    return float((plan * costs).sum()), plan
```

hungarian.py

```
import numpy as np


EPS, MAX_ITER = 1e-9, 200



def max_matching(M, n):
    match = [-1] * n


    def try_assign(i, visited):
        for j in range(n):
            if abs(M[i, j]) < EPS and not visited[j]:
                visited[j] = True
                if match[j] == -1 or try_assign(match[j], visited):
                    match[j] = i
                    return True
        return False


    for i in range(n):
        try_assign(i, [False] * n)
    return match



def hungarian(cost, minimize=True):
    orig = np.array(cost, float)
    rows, cols = orig.shape
    n = max(rows, cols)
    M = np.zeros((n, n))
    M[:rows, :cols] = orig
    if not minimize:
        M = M.max() - M
    M -= M.min(axis=1, keepdims=True)
    M -= M.min(axis=0, keepdims=True)
    for _ in range(MAX_ITER):
        match = max_matching(M, n)
        if sum(j != -1 for j in match) == n:
            break
        assigned_rows = {match[j] for j in range(n) if match[j] != -1}
        marked_rows = {i for i in range(n) if i not in assigned_rows}
        marked_cols = set()
        changed = True
```

```
        while changed:
            changed = False
            for i in list(marked_rows):
                for j in range(n):
                    if abs(M[i, j]) < EPS and j not in marked_cols:
                        marked_cols.add(j)
                        changed = True
            for j in list(marked_cols):
                if match[j] != -1 and match[j] not in marked_rows:
                    marked_rows.add(match[j])
                    changed = True
        unc_rows = [i for i in range(n) if i in marked_rows]
        unc_cols = [j for j in range(n) if j not in marked_cols]
        delta = M[np.ix_(unc_rows, unc_cols)].min()
        for i in unc_rows:
            M[i, :] -= delta
        for j in sorted(marked_cols):
            M[:, j] += delta
    match = max_matching(M, n)
    pairs = [(match[j], j) for j in range(n)
             if match[j] != -1 and match[j] < rows and j < cols]
    return float(sum(orig[i, j] for i, j in pairs)), pairs
```

dp_knapsack.py

```
def knapsack(weights, values, capacity):
    n = len(weights)
    dp = [[0.0] * (capacity + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        w_i, v_i = weights[i - 1], values[i - 1]
        for w in range(capacity + 1):
            dp[i][w] = dp[i - 1][w]
            if w_i <= w:
                dp[i][w] = max(dp[i][w], dp[i - 1][w - w_i] + v_i)
    selected, w = [], capacity
    for i in range(n, 0, -1):
        if dp[i][w] != dp[i - 1][w]:
            selected.append(i - 1)
            w -= weights[i - 1]
    selected.reverse()
    return dp[n][capacity], selected
```

ДОДАТОК Б

“Вебсистема підтримки прийняття рішень на основі методів оптимізації”

УКР.НТУУ “КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР_23

Копія публікації тез доповіді

Аркушів 3

Київ – 2026

SECTION 17.
SYSTEM ANALYSIS, MODELING AND OPTIMIZATION

Скакодуб Світлана Олександрівна
здобувач вищої освіти Навчально-наукового інституту атомної та теплової енергетики
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Україна

Науковий керівник: Сліпченко Володимир Георгійович
доктор наук, професор, кафедра цифрових технологій в енергетиці
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Україна

ВЕБСИСТЕМА ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ НА ОСНОВІ МЕТОДІВ ОПТИМІЗАЦІЇ

Прийняття рішень про розподіл обмежених ресурсів є типовим класом задач, що щодня постають у плануванні виробничої, постачальницько-збутової та логістичної діяльності підприємств. Математичною основою для їх розв'язання слугує апарат дослідження операцій — методи лінійного й цілочислового програмування, методи розв'язання транспортних задач, задач про призначення та про рюкзак [1, 2]. Попри те, що формальний апарат цих методів добре формалізовано, ефективне практичне використання класичних інструментів — алгебраїчних мов моделювання та обчислювальних солверів — потребує від користувача глибокої фахової підготовки в дослідженні операцій і досвіду програмування. Прозорі веб-калькулятори знімають цей бар'єр, але обмежуються одним методом і одним типом задачі та не передбачають єдиної історії розв'язань і аналізу чутливості [6]. Розробка вебсистеми, яка поєднує широке охоплення класів задач, автоматичну рекомендацію методу й покрокову візуалізацію обчислень, є тому актуальною й практично виправданою.

Аналіз наявних програмних засобів виявляє три класи інструментів [6]. Перший — двошарові платформи моделювання (алгебраїчні мови AMPL, GAMS, Pyomo, PuLP у поєднанні з солверами Gurobi, CPLEX, CBC, GLPK): найвища швидкодія на задачах великої розмірності, проте висока вимога до кваліфікації користувача й мінімальна прозорість обчислень. Другий — веб-калькулятори окремих методів: простота доступу без установлення ПЗ, але тематична вузькість, відсутність аналізу чутливості та єдиної історії. Третій — навчальні системи, що демонструють кроки алгоритму, проте кожна

покриває лише одну тему й не зберігає персональної історії. Жоден клас не покриває комбінацію вимог — орієнтацію на користувача без фахової підготовки, широке охоплення класів задач і простоту веб-доступу одночасно.

Мета роботи — розробка вебсистеми підтримки прийняття рішень на основі методів оптимізації для автоматизованого формулювання й розв'язання задач лінійного та цілочислового програмування, транспортних задач, задач про призначення та про рюкзак з покроковою візуалізацією обчислень і аналізом чутливості, орієнтованої на користувача без спеціальної підготовки в дослідженні операцій [3].

Запропонована система реалізована як трирівневий веб-застосунок: клієнтська частина на Next.js і TypeScript, серверний застосунок на Python з фреймворком FastAPI та реляційна СКБД PostgreSQL. Обчислювальне ядро побудовано на NumPy без використання зовнішніх готових солверів — це забезпечує повний контроль над збором проміжних ітерацій, необхідних для покрокової візуалізації. Солвери реалізовані за патерном «Стратегія» з єдиним інтерфейсом, що спрощує розширення системи новими методами.

Для розв'язання обрано п'ять класичних методів, спільною властивістю яких є придатність до покрокової візуалізації, узгодженої з прикладною сутністю кожного класу: симплекс-метод — для задач лінійного програмування з неперервними змінними [4]; метод гілок і меж — для змішаного цілочислового програмування [1]; метод потенціалів (MODI) — для транспортних задач [1, 2]; угорський алгоритм Куна–Мункреса — для задач про призначення [5]; класичний двовимірний варіант динамічного програмування — для задачі про рюкзак 0/1 [1]. Ключовим елементом доступності є сервіс автоматичної класифікації типу задачі та рекомендації методу: користувач формалізує задачу через шаблонну форму або довільну постановку, а система сама добирає метод і надає текстове обґрунтування вибору українською мовою.

Розроблене програмне забезпечення підтримує реєстрацію з ізоляцією даних користувачів, покрокову візуалізацію процесу обчислень, специфічну для кожного методу (симплекс-таблиці, інтерактивне дерево гілкування, графічну побудову допустимої області), аналіз чутливості для лінійних задач, ведення персональної історії з фільтрацією та what-if-варіантами, а також генерацію PDF-звітів. Усі етапи — від формулювання задачі до отримання звіту — виконуються в межах єдиного веб-інтерфейсу без потреби у спеціалізованому програмному забезпеченні з боку користувача.

Коректність роботи підтверджено автоматизованим тестуванням в обсязі 109 тест-кейсів, що покривають усі обчислювальні модулі, REST API, цикл

автентифікації та формування звітів; кожен солвер пройшов крос-верифікацію проти еталонних реалізацій бібліотеки SciPy. Емпірично підтверджені діапазони розмірності задач відповідають навчальному та демонстраційному призначенню системи.

Висновки. Розроблено вебсистему підтримки прийняття рішень, що поєднує покрокову візуалізацію навчальних систем, широке охоплення класів задач платформ моделювання й простоту веб-доступу онлайн-калькуляторів, доповнюючи їх автоматичною класифікацією типу задачі та аналізом чутливості. Запропонований підхід знижує бар'єр входу до класичних методів оптимізації для користувачів без фахової підготовки в дослідженні операцій. Перспективи розвитку пов'язані з реалізацією задачі про розкрій методом генерації стовпців, інтеграцією методів багатоцільової оптимізації для пошуку Парето-оптимальних розв'язань і створенням модуля порівняльного бенчмаркінгу методів за розмірністю задачі.

Список використаних джерел:

1. Hillier F. S., Lieberman G. J. Introduction to Operations Research. 11th ed. New York : McGraw-Hill Education, 2021. 1048 p.
2. Григорків В. С., Григорків М. В., Ярошенко О. І. Оптимізаційні методи та моделі: навч. посіб. Чернівці : ЧНУ, 2020. 200 с.
3. Turban E., Aronson J. E., Liang T.-P. Decision Support Systems and Intelligent Systems. 7th ed. Upper Saddle River : Pearson Prentice Hall, 2005. 867 p.
4. Vanderbei R. J. Linear Programming : Foundations and Extensions. 5th ed. Cham : Springer, 2020. 471 p.
5. Сікора Я. Б., Щехорський А. Й., Якимчук Б. Л. Методи оптимізації та дослідження операцій : навч. посіб. Житомир : ЖДУ ім. І. Франка, 2019. 164 с.
6. Dastres R., Soori M. Advances in Web-Based Decision Support Systems. Journal of Information Systems and Telecommunication. 2022. Vol. 19, № 1. P. 56–65.
7. Bean R., Khan H. Using solar and load predictions in battery scheduling at the residential level. 2018. 6 p. URL: https://arxiv.org/abs/1810.11178.