Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«____»_____2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИВЧЕННЯ ЦІЛОЧИСЛОВОГО ПРОГРАМУВАННЯ У КУРСІ МЕТОДІВ ОПТИМІЗАЦІЇ ТА ДОСЛІДЖЕННЯ ОПЕРАЦІЙ»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавр

Виконавець роботи Дрижирук Павло Миколайович

_____«__»_____2026р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Парфьонова Тетяна Олександрівна

_____«__»_____2026р.
(підпис)

Рецензент _____

ПОЛТАВА 2026

РЕФЕРАТ

Записка: 53 с., 11 рис., 6 табл., 1 додаток, 15 джерел.

ЦІЛОЧИСЛОВЕ ПРОГРАМУВАННЯ, СИМПЛЕКС-МЕТОД, МЕТОД ГІЛОК І МЕЖ, ТРАНСПОРТНА ЗАДАЧА, НАВЧАЛЬНИЙ ТРЕНАЖЕР, ВЕБЗАСТОСУНОК, ПОКРОКОВА ВІЗУАЛІЗАЦІЯ.

Об’єкт розробки - процес навчання студентів спеціальності «Комп’ютерні науки» розв’язуванню задач цілочислового програмування у межах дисципліни «Методи оптимізації та дослідження операцій».

Мета роботи - проєктування та програмна реалізація навчального вебтренажера для опанування методів розв’язування задач лінійного та цілочислового програмування з покроковою візуалізацією та перевіркою дій студента.

Методи дослідження - методи лінійного та цілочислового програмування (симплекс-метод, метод гілок і меж, метод потенціалів для транспортної задачі) для побудови алгоритмічного ядра; методи об’єктно-орієнтованого аналізу та UML-моделювання для проєктування архітектури; методи модульного й інтеграційного тестування для оцінки якості реалізації.

Результати та їх новизна. Удосконалено підхід до навчання цілочисловому програмуванню шляхом поєднання в єдиному вебсередовищі теоретичного матеріалу, бібліотеки задач, генератора та покрокового інтерактивного розв’язання з автоматичною перевіркою кожної дії студента і деталізованим показом самих обчислень. Здійснено програмну реалізацію тренажера та розгорнуто його на хмарних платформах для цілодобового доступу.

Рекомендації щодо використання - тренажер може застосовуватися в дистанційному курсі «Методи оптимізації та дослідження операцій» для самостійної роботи студентів, як інструмент лабораторних занять і як засіб самоконтролю.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, скорочення, терміни	**Пояснення**
ЗЛП	задача лінійного програмування
ЛП	лінійне програмування
МГМ	метод гілок і меж
ПЗ	програмне забезпечення
ЦП	цілочислове програмування
ЦФ	цільова функція
API	Application Programming Interface — прикладний програмний інтерфейс
CDN	Content Delivery Network — мережа доставки контенту
CORS	Cross-Origin Resource Sharing — механізм крос-доменних запитів
HTTP	HyperText Transfer Protocol — протокол передавання гіпертексту
JSON	JavaScript Object Notation — текстовий формат обміну даними
MODI	Modified Distribution — метод потенціалів
REST	Representational State Transfer — архітектурний стиль вебсервісів
SPA	Single Page Application — односторінковий вебзастосунок
UI	User Interface — інтерфейс користувача
UML	Unified Modeling Language — уніфікована мова моделювання
WSGI	Web Server Gateway Interface — інтерфейс вебсервера з Python-застосунком

ВСТУП

Актуальність теми. Сучасний розвиток економіки, виробництва, логістики й управління безпосередньо пов’язаний із застосуванням оптимізаційних математичних моделей. Особливе місце серед них посідають задачі цілочислового програмування, які описують ситуації, коли змінні мають набувати лише цілих значень: планування випуску дискретної продукції, розподіл ресурсів, маршрутизація, призначення виконавців на завдання, формування інвестиційного портфеля. У загальному випадку такі задачі є NP-складними, тому студент має розуміти не лише математичну постановку, а й внутрішню логіку алгоритмів їх розв’язання.

У підготовці фахівців спеціальності 122 «Комп’ютерні науки» дисципліна «Методи оптимізації та дослідження операцій» належить до базових, і її засвоєння потребує значного обсягу обчислювальної практики. Розв’язування задач симплекс-методом, методом гілок і меж та методом потенціалів «вручну» є рутинним і трудомістким, а помилка на одному з проміжних кроків призводить до хибної відповіді й суттєвих втрат часу. Промислові системи (CPLEX, Gurobi, LINDO) знімають проблему обчислень, але приховують хід алгоритму, перетворюючи його на «чорну скриньку». Для навчання ж потрібен інструмент, який показує кожен крок, дозволяє студентові виконати цей крок самостійно й одразу отримати оцінку правильності з підказкою. Розробка такого програмного забезпечення відповідає тенденціям цифровізації освіти та підвищення якості підготовки фахівців, що й зумовлює актуальність обраної теми.

Мета роботи - спроєктувати й програмно реалізувати навчальний вебтренажер для опанування методів розв’язування задач лінійного та цілочислового програмування в межах курсу «Методи оптимізації та дослідження операцій».

Задачі роботи:

1) проаналізувати предметну область - задачі лінійного та цілочислового програмування і методи їх розв’язання - та сформулювати вимоги до тренажера;

2) оглянути наявні програмні рішення, виявити їхні переваги й обмеження стосовно навчальних цілей;

3) дослідити симплекс-метод, метод гілок і меж та метод потенціалів як алгоритмічну основу тренажера;

4) спроєктувати клієнт-серверну архітектуру системи, побудувати UML-діаграми та обґрунтувати вибір технологічного стека;

5) реалізувати програмне забезпечення з підтримкою чотирьох типів задач і трьох режимів навчання;

6) перевірити коректність роботи тренажера та підготувати інструкцію користувача.

Об’єкт дослідження - процес навчання студентів методів розв’язування задач цілочислового програмування у вищій школі.

Предмет дослідження - моделі, алгоритми та програмні засоби, що забезпечують інтерактивне навчання цілочисловому програмуванню з покроковою візуалізацією та перевіркою дій студента.

Методи дослідження. У роботі використано методи теорії оптимізації (лінійне та цілочислове програмування, симплекс-метод, метод гілок і меж, метод потенціалів) - для побудови алгоритмічного ядра; методи об’єктно-орієнтованого аналізу й проєктування та UML-моделювання - для розробки архітектури; методи модульного, інтеграційного та функціонального тестування - для оцінки якості програмної реалізації.

Наукова новизна та практичне значення. Удосконалено підхід до організації навчання цілочисловому програмуванню за рахунок поєднання в єдиному вебсередовищі: теоретичного матеріалу за кожним методом; бібліотеки типових задач і генератора нових задач; трирівневої прогресії складності - від перегляду готового розв’язку через інтерактивний вибір дій

із валідацією до самостійного перерахування таблиці в «сліпому» режимі; автоматичної перевірки кожної дії студента з підсвічуванням конкретних помилок і деталізованим показом самих обчислень. На відміну від наявних онлайн-калькуляторів, тренажер українськомовний, охоплює три методи одночасно й перевіряє кроки, а не лише показує відповідь.

Практична цінність. Розроблений тренажер придатний до безпосереднього використання в дистанційному курсі «Методи оптимізації та дослідження операцій» для самостійної підготовки студентів, як інструмент проведення лабораторних занять і як засіб самоконтролю. Розгортання на безкоштовних хмарних платформах забезпечує доступ до нього цілодобово без обслуговування з боку розробника.

Структура та обсяг роботи. Робота складається зі вступу, чотирьох розділів, висновків, рекомендацій, списку використаних джерел та додатків. У першому розділі наведено постановку задачі та математичну модель; у другому - огляд аналогів; у третьому - теорію методів, проєктування архітектури та обґрунтування технологій; у четвертому - опис реалізації, перевірку роботи та інструкцію користувача.

1. ПОСТАНОВКА ЗАДАЧІ

На початковому етапі дослідження необхідно визначити змістовну постановку задачі та сформулювати основні вимоги до тренажера, що забезпечать його ефективне функціонування. Дисципліна «Методи оптимізації та дослідження операцій» передбачає опанування студентами точних методів розв’язування оптимізаційних задач

- насамперед симплекс-методу для задач лінійного програмування, методу гілок і меж для задач цілочислового програмування та методу потенціалів для транспортної задачі. Ці методи є ітераційними: розв’язок будується послідовністю однотипних кроків, на кожному з яких студент має прийняти рішення (обрати ведучий елемент, змінну для розгалуження, клітинку для введення в базис) і виконати перерахунок. Засвоєння таких алгоритмів неможливе лише через читання теорії - воно потребує багаторазового самостійного відпрацювання, де помилка на одному кроці одразу спотворює весь подальший розв’язок [3, 4].

Традиційні засоби навчання цього не забезпечують. Промислові та бібліотечні розв’язувачі видають лише готову відповідь, приховуючи хід обчислень; онлайн-калькулятори в кращому разі показують кроки, але не дозволяють студентові виконати крок самостійно й не перевіряють його дій. Тому виникає потреба у спеціалізованому навчальному програмному забезпеченні - тренажері, який поєднує показ алгоритму, самостійне виконання кроків студентом та негайний зворотний зв’язок.

Дано: програма дисципліни «Методи оптимізації та дослідження операцій»; перелік методів, що становлять навчальне ядро курсу, - симплекс-метод, метод гілок і меж, метод потенціалів (транспортна задача); вимоги до програмного засобу - вебдоступ, українськомовний інтерфейс, можливість самостійного виконання кроків із перевіркою.

Потрібно розробити вебзастосунок (навчальний тренажер), який задовольняє наведеним нижче функціональним вимогам (таблиця 1.1).

Таблиця 1.1 - Функціональні вимоги до тренажера

Позначення	Зміст вимоги
Ф1	надавати теоретичну довідку (математична модель, алгоритм, умова оптимальності) за кожним методом
Ф2	містити бібліотеку готових типових задач із можливістю вибору
Ф3	генерувати нові випадкові задачі заданої розмірності
Ф4	дозволяти введення власної задачі у структурованій формі
Ф5	підтримувати три режими роботи: перегляд усіх кроків, покроковий інтерактивний прохід, «сліпий» режим самостійного перерахунку таблиці
Ф6	на кожному кроці перевіряти дію студента (вибір ведучого стовпця/рядка, змінної розгалуження, вхідної клітинки) та підсвічувати помилки з підказкою
Ф7	показувати деталізовані обчислення кроку — нормування ведучого рядка, виключення за Гаусом, перерозподіл за циклом
Ф8	відображати еталонний покроковий розв’язок задачі для аналізу
Ф9	зберігати статистику навчання (кількість спроб, помилок, підказок) локально та надавати можливість її скидання
Ф10	надавати окремий тренувальний блок для самостійного обчислення потенціалів транспортної задачі

Нефункціональні вимоги: доступ через будь-який сучасний браузер без встановлення додаткового програмного забезпечення; українськомовний інтерфейс; час відгуку на перевірку кроку та розв’язання задачі - не більше однієї секунди для навчальних розмірностей; адаптивність інтерфейсу до різних роздільних здатностей; відсутність стану на сервері (кожен запит самодостатній), що спрощує розгортання та масштабування; побудова на безкоштовному й відкритому технологічному стеку з можливістю розгортання на хмарних платформах.

Обмеження. Тренажер орієнтовано на навчальні задачі невеликої розмірності - типово до 5-6 змінних та обмежень для симплекс-методу й методу гілок і меж і до 5×5 для транспортної задачі. Це покриває обсяги типових контрольних і лабораторних робіт та водночас забезпечує миттєвий відгук в інтерактивному режимі. Транспортна задача розв’язується у

збалансованій постановці; незбалансовані задачі балансуються автоматично введенням фіктивного постачальника або споживача.

Для формалізації поставленої задачі доцільно побудувати математичну модель та дослідити її основні характеристики, які визначають особливості функціонування тренажера. Тренажер працює з двома спорідненими класами оптимізаційних задач - задачами лінійного (цілочислового) програмування у загальній формі та транспортною задачею.

Задача лінійного та цілочислового програмування. Загальна математична модель задачі лінійного програмування полягає у знаходженні екстремуму лінійної цільової функції за лінійних обмежень:

$$Z = c_1 x_1 + c_2 x_2 + \ldots + c_j x_j \rightarrow max\ (min), \tag{1.1}$$

за умов

$$a_{i1} x_1 + a_{i2} x_2 + \ldots + a_{ij} x_j \leq b_i,\ i = 1, \ldots, m, \tag{1.2}$$

$$x_j \geq 0,\ j = 1, \ldots, n, \tag{1.3}$$

де c_j - коефіцієнти цільової функції (питомий прибуток, вартість або інший показник одиниці змінної); a_{ij} — норми витрат і-го ресурсу на одиницю j-ї змінної; b_i - наявний обсяг і-го ресурсу; x_j - шукані змінні; Z - значення цільової функції; n - кількість змінних; m - кількість обмежень.

Задача цілочислового програмування отримується з (1.1)–(1.3) додаванням вимоги цілочисловості частини або всіх змінних:

$$x_j \in Z,\ j \in J \subseteq \{1, \ldots, n\}. \tag{1.4}$$

Якщо J = {1, …, n}, задача є повністю цілочисловою; якщо J ⊊ {1, …, n} - змішаною. Принципова відмінність між (1.1)–(1.3) і (1.1)–(1.4) визначає й різні методи розв’язання. У задачі лінійного програмування множина допустимих розв’язків - опуклий багатогранник, і оптимум завжди досягається в одній з його вершин, тому застосовний симплекс-метод. У задачі цілочислового програмування множина допустимих розв’язків дискретна; її оптимум, як правило, не збігається з вершиною багатогранника лінійної релаксації, а просте округлення координат може вивести точку за

межі допустимої області. Такі задачі у загальному випадку належать до класу NP-складних, тому для них застосовують метод гілок і меж, що поєднує розв'язання лінійних релаксацій із відсіканням безперспективних гілок [1, 5].

Приклад 1.1. Підприємство виготовляє два види виробів - A і B. Прибуток від одиниці виробу A становить 3 ум. од., виробу B - 4 ум. од. На виготовлення одиниці A витрачається 1 одиниця матеріалу та 3 години роботи устаткування, на одиницю B - 2 одиниці матеріалу та 2 години. Запас матеріалу - 14 одиниць, ресурс часу - 18 годин. Потрібно визначити план виробництва, що максимізує прибуток. Позначивши через x_1 і x_2 кількість виробів A і B, отримуємо модель:

$$Z = 3x_1 + 4x_2 \rightarrow max, \tag{1.5}$$

$$x_1 + 2x_2 \leq 14, \tag{1.6}$$

$$3x_1 + 2x_2 \leq 18, \tag{1.7}$$

$$x_1, x_2 \geq 0 \text{ (за потреби цілочисловості - } x_1, x_2 \in Z). \tag{1.8}$$

Транспортна задача. Окремий клас становить транспортна задача - знаходження плану перевезень однорідного вантажу від m постачальників із запасами a_i до n споживачів із потребами b_j за мінімальної сумарної вартості:

$$F = \Sigma_i \Sigma_j c_{ij} x_{ij} \rightarrow min, \; i = 1, \ldots, m, \; j = 1, \ldots, n, \tag{1.9}$$

за умов

$$\Sigma_j x_{ij} = a_i, \; i = 1, \ldots, m, \tag{1.10}$$

$$\Sigma_i x_{ij} = b_j, \; j = 1, \ldots, n, \tag{1.11}$$

$$x_{ij} \geq 0, \tag{1.12}$$

де c_{ij} - вартість перевезення одиниці вантажу від i-го постачальника до j-го споживача; x_{ij} - обсяг перевезення цим маршрутом. Задача має розв'язок за умови балансу запасів і потреб:

$$\Sigma_i a_i = \Sigma_j b_j. \tag{1.13}$$

Якщо умова (1.13) не виконується, задачу балансують уведенням фіктивного постачальника або споживача з нульовими тарифами. Опорний (базисний) план транспортної задачі містить рівно $m + n - 1$ заповнену клітинку; саме на цій властивості ґрунтуються метод північно-західного кута для побудови початкового плану та метод потенціалів для його оптимізації.

Характеристики моделей. Усі три задачі лінійні за цільовою функцією та обмеженнями, що забезпечує застосовність точних ітераційних методів і робить їх придатними для покрокової демонстрації. Розмірність моделей у тренажері навмисно обмежена навчальними значеннями (підрозділ 1.1): це гарантує доступність повного покрокового протоколу розв’язання для відображення на екрані та час обчислень у межах частки секунди. Спільна лінійна природа задач дозволяє реалізувати єдину форму введення умови для симплекс-методу й методу гілок і меж та окрему - для транспортної задачі, що відображено в архітектурі застосунку (розділ 3).

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд існуючих програмних рішень для розв’язання задач оптимізації

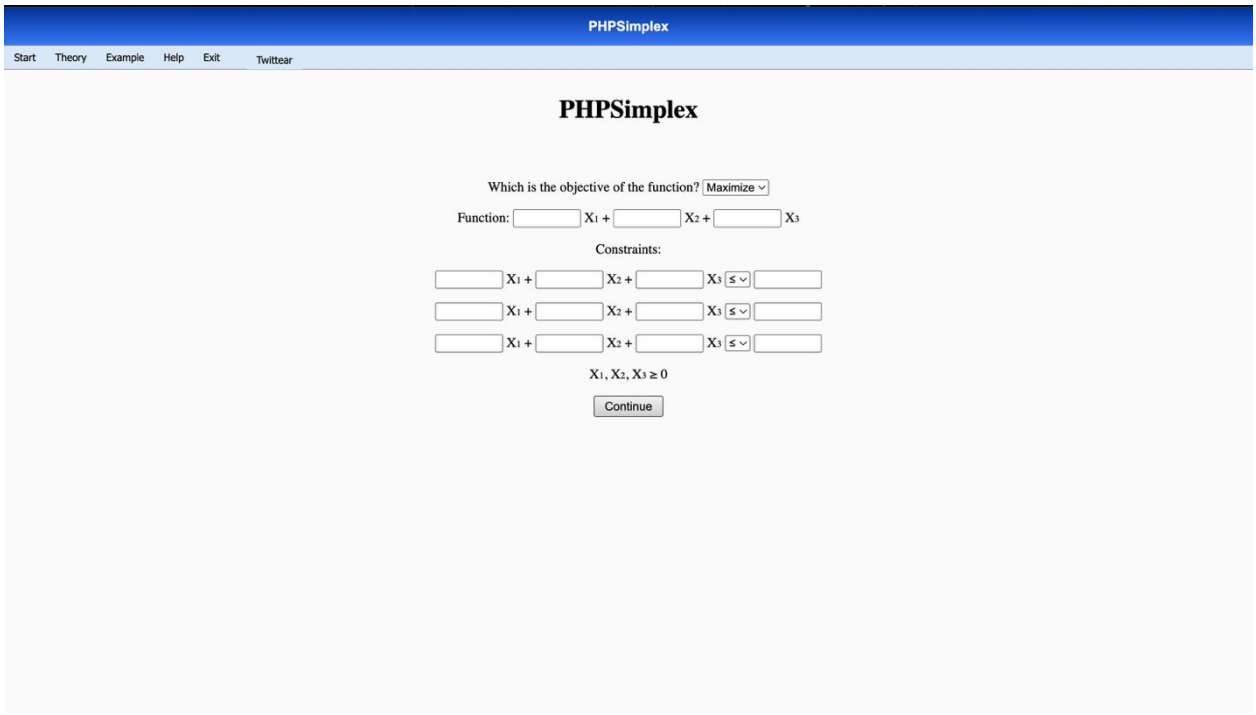
На сьогодні існує широкий спектр програмних засобів, які тією чи іншою мірою можна застосувати для розв’язування задач лінійного та цілочислового програмування. Для аналізу їхньої придатності саме до навчальних цілей доцільно поділити ці засоби на три категорії: промислові оптимізаційні розв’язувачі, відкриті програмні бібліотеки та навчальні онлайн-сервіси.

Промислові розв’язувачі. Лідерами галузі вважаються IBM ILOG CPLEX Optimization Studio [7] та Gurobi Optimizer [8]. Вони реалізують десятки високооптимізованих алгоритмів - прямий і двоїстий симплекс-метод, метод внутрішніх точок, метод гілок і меж із сучасними евристиками, відсікаючими площинами та паралельними обчисленнями. Їхня сильна сторона - здатність розв’язувати задачі з мільйонами змінних за прийнятний час. Пакет LINDO/LINGO історично орієнтований на навчальні установи й невеликі задачі, має зрозумілий синтаксис, наближений до математичної нотації. MATLAB Optimization Toolbox містить функції linprog та intlinprog, що реалізують відповідно симплекс-метод і розв’язування змішаних цілочислових задач, і добре інтегрований з рештою інструментів MATLAB. Спільною перевагою цих засобів є потужність і надійність; спільним недоліком для навчання - закритість ходу обчислень: студент бачить лише вхідні дані та готову відповідь, тоді як проміжні таблиці й рішення алгоритму приховано. Більшість із них також є комерційними.

Відкриті програмні бібліотеки. Для мови Python розроблено бібліотеки, що дозволяють зручно описувати й розв’язувати оптимізаційні задачі: PuLP [9] - високорівнева обгортка для опису задач лінійного й цілочислового програмування, яка працює з безкоштовними розв’язувачами CBC і GLPK; SciPy із модулями scipy.optimize.linprog та scipy.optimize.milp

[10] - реалізації симплекс-методу й змішаного цілочислового програмування у складі наукової бібліотеки; Google OR-Tools - потужний набір алгоритмів від маршрутизації до календарного планування. Ці бібліотеки безкоштовні, кросплатформні й активно розвиваються. Проте за призначенням вони є саме розв’язувачами: повертають готову відповідь, не показуючи проміжних кроків, і не передбачають самостійного виконання кроку користувачем із подальшою перевіркою.

Навчальні онлайн-сервіси. Існують вебресурси (PHPSimplex, різноманітні Simplex Tableau Calculator, онлайн-калькулятори транспортної задачі), які надають можливість покрокового розв’язання - насамперед симплекс-методом. Вони безкоштовні й не потребують встановлення. Однак їхні обмеження для українського навчального процесу суттєві: переважна більшість не має українськомовного інтерфейсу та теоретичної підтримки; зазвичай реалізовано лише один метод; студент може спостерігати кроки, але не може виконати крок самостійно й отримати оцінку правильності; відсутні бібліотека типових задач і генератор нових.



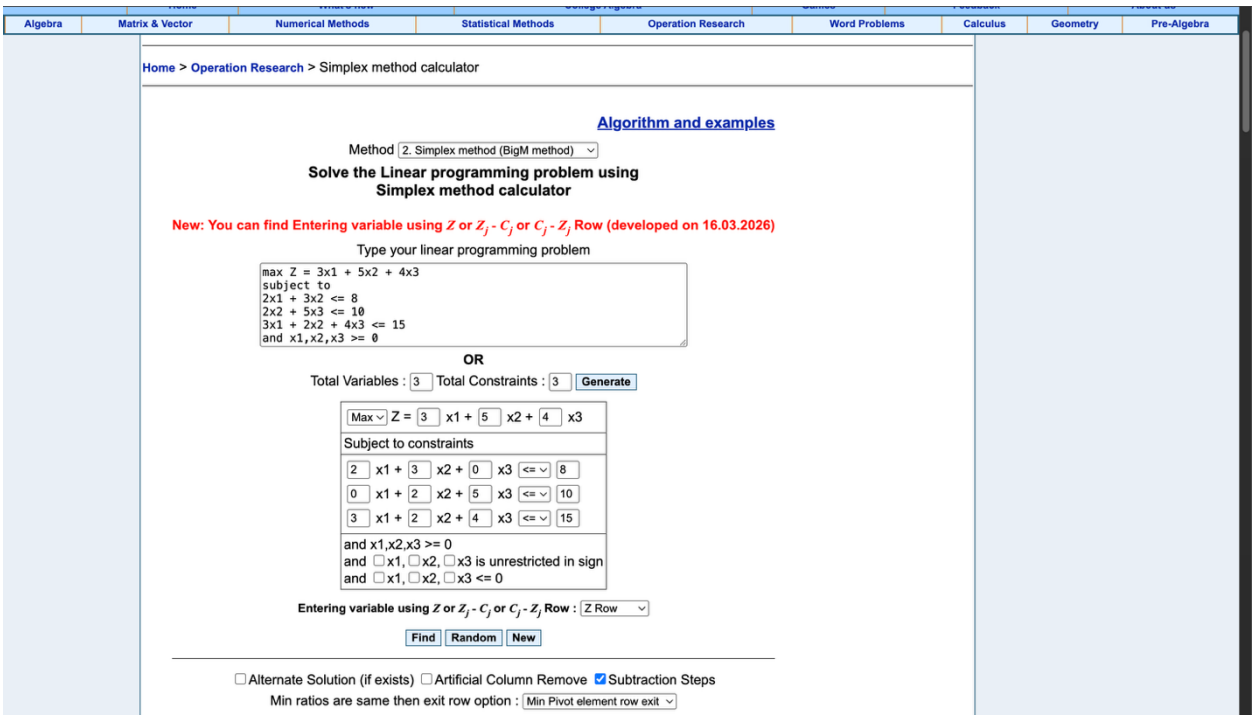


Рисунок 2.1-2.2 - Приклад інтерфейсу наявного онлайн-калькулятора симплекс-методу

Узагальнену порівняльну характеристику розглянутих рішень за критеріями придатності для навчальних цілей наведено в таблиці 2.1.

Таблиця 2.1 - Порівняння існуючих програмних рішень

Рішення	**Покроковий показ**	**Кількість методів**	**Україномовний інтерфейс**	**Безкоштовність**	**Перевірка кроку**
CPLEX	ні	так	ні	ні	ні
Gurobi	ні	так	ні	ні	ні
LINDO/LINGO	частково	так	ні	частково	ні
MATLAB Opt. Toolbox	ні	так	ні	ні	ні
PuLP / SciPy / OR-Tools	ні	так	ні	так	ні
Онлайн-калькулятори	так	ні	ні	так	ні
Розроблюваний тренажер	**так**	**так**	**так**	**так**	**так**

* Під «кількома методами» розуміється одночасна підтримка симплекс-методу, методу гілок і меж та транспортної задачі.

2.2 Недоліки оглянутих рішень та обґрунтування актуальності розробки

Аналіз таблиці 2.1 засвідчує, що жодне з оглянутих рішень не задовольняє одночасно всім вимогам, які постають у навчальному процесі. Узагальнено їхні недоліки стосовно навчальних цілей зводяться до такого:

- промислові розв’язувачі та бібліотеки приховують хід алгоритму, перетворюючи його на «чорну скриньку»: студент не бачить ані симплекс-таблиць, ані дерева гілок і меж, ані циклу перерозподілу транспортної задачі, а отже не може простежити логіку методу;

- жоден із засобів не дозволяє студентові самостійно виконати крок алгоритму (обрати ведучий елемент, змінну для розгалуження, вхідну клітинку чи перерахувати таблицю) і не перевіряє правильність цієї дії — а саме така активна практика є ключовою для засвоєння ітераційних методів;

- українськомовних навчальних рішень практично немає, що створює додатковий бар’єр для студентів;

- наявні навчальні онлайн-калькулятори обмежені, як правило, одним методом, не містять теоретичної довідки, бібліотеки типових задач і генератора нових, а тому не покривають програму курсу;

- більшість потужних засобів є комерційними, що ускладнює їх масове використання у самостійній роботі студентів.

Виявлені прогалини безпосередньо визначають вимоги до розроблюваного тренажера (підрозділ 1.1). На противагу оглянутим рішенням, він має поєднувати в єдиному безкоштовному вебсередовищі: покрокову візуалізацію всіх трьох методів із показом самих обчислень (вимоги Ф5, Ф7); інтерактивну перевірку кожної дії студента з підсвічуванням помилок і підказками (Ф6); теоретичну довідку, бібліотеку готових задач і генератор нових (Ф1–Ф4); українськомовний інтерфейс. Саме

така комбінація переводить програму з категорії «розв’язувач» до категорії «навчальний інструмент» і відрізняє її від усіх розглянутих аналогів.

Розробка такого програмного забезпечення відповідає сучасним тенденціям цифровізації освіти, розвитку дистанційного навчання та підвищення якості підготовки фахівців у галузі комп’ютерних наук. Поєднання теоретичної, практичної та контрольної складових в одному доступному цілодобово вебзастосунку дозволяє студентові навчатися у власному темпі, а викладачеві - використовувати тренажер під час лабораторних занять і для самоконтролю студентів. Це й обґрунтовує актуальність та доцільність розробки навчального тренажера з методів цілочислового програмування.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Математичні методи розв'язання задач лінійного та цілочислового програмування

Алгоритмічним ядром тренажера є три класичні точні методи: симплекс-метод для задач лінійного програмування, метод гілок і меж для задач цілочислового програмування та метод потенціалів для транспортної задачі. Усі три є ітераційними й мають прозору покрокову структуру, що робить їх придатними як для автоматичного розв'язання, так і для покрокової навчальної демонстрації. Нижче розглянуто принцип роботи кожного з них у тому вигляді, в якому його реалізовано в програмі.

Симплекс-метод. Симплекс-метод, запропонований Дж. Данцигом, є основним алгоритмом розв'язання задач лінійного програмування [4]. Геометрично він стартує з однієї вершини багатогранника допустимих розв'язків і послідовно переходить до сусідньої вершини з кращим значенням цільової функції, доки не досягне оптимуму [1, 2].

Для застосування методу задачу зводять до канонічної форми: кожне обмеження-нерівність виду «≤» перетворюють на рівність уведенням балансової (slack) змінної $s_i \geq 0$:

$$a_{i1}\ x_1\ + \ldots + a_i\square x\square + s_i = b_i,\ i = 1, \ldots, m. \quad (3.1)$$

Сукупність коефіцієнтів зводять у симплекс-таблицю, рядки якої відповідають базисним змінним, а останній рядок містить оцінки (зведені вартості) небазисних змінних

$$\Delta\square = c\square - z\square, \quad (3.2)$$

де $z\square$ - внесок j-ї змінної через поточний базис. Початковим базисом слугують балансові змінні $\{s_1\ , \ldots, s\square\}$.

Алгоритм виконує таку послідовність кроків:

1. сформувати початкову симплекс-таблицю та базис;
2. перевірити умову оптимальності: для задачі максимізації план оптимальний, якщо всі $\Delta\square \leq 0$ (для мінімізації - $\Delta\square \geq 0$); якщо так - перейти до кроку 6;

3. обрати ведучий стовпець за правилом Данціга - стовпець із найбільшою за модулем оцінкою, що покращує ціль (для максимізації - найбільша додатна Δ□); відповідна змінна вводиться в базис;
4. обрати ведучий рядок за правилом мінімального відношення: серед рядків з додатним елементом ведучого стовпця a_i□ > 0 обчислити $\theta_i = b_i / a_i$□ і взяти рядок із найменшим θ; якщо всіх a_i□ ≤ 0 - цільова функція необмежена;

   $\theta_i = b_i / a_i$□, a_i□ > 0, (3.3)
5. виконати pivot-перетворення (Гаусове виключення): поділити ведучий рядок на ведучий елемент, а від решти рядків відняти ведучий рядок, помножений на відповідний коефіцієнт, щоб у ведучому стовпці лишилася одиниця у ведучому рядку; оновити базис і повернутися до кроку 2;
6. зчитати значення базисних змінних (небазисні дорівнюють нулю) та обчислити оптимальне значення цільової функції.

У програмі реалізовано саме двофазний табличний симплекс-метод із правилом Данціга для вибору ведучого стовпця та правилом мінімального відношення для вибору ведучого рядка; на кожній ітерації зберігається «знімок» таблиці, що дозволяє відобразити весь хід розв'язання покроково.

Метод гілок і меж. Метод гілок і меж - точний метод розв'язання задач цілочислового програмування, що поєднує впорядкований перебір із відсіканням безперспективних варіантів [5]. Його ідея - багаторазове розв'язання лінійних релаксацій вихідної задачі з поступовим додаванням обмежень, що «відрізають» нецілі розв'язки.

Алгоритм будує дерево пошуку, корінь якого - лінійна релаксація задачі цілочислового програмування (умову цілочисловості тимчасово відкинуто). Розв'язавши релаксацію симплекс-методом, аналізують результат за такими правилами:

1. якщо область допустимих розв'язків порожня - вузол недопустимий і відкидається;

2. якщо значення релаксації не може покращити поточний рекорд (для максимізації - оцінка вузла не перевищує найкращого знайденого цілого розв'язку) - вузол відсікається;
3. якщо отриманий розв'язок цілочисловий - він є кандидатом на оптимум, і рекорд оновлюється;
4. інакше виконується розгалуження: обирається змінна x□ із дробовим значення x□*, і вузол ділиться на два нащадки додаванням обмежень

   x□ ≤ ⌊x□*⌋ та x□ ≥ ⌈x□*⌉. (3.4)

Цей поділ гарантує, що жоден цілий розв'язок не буде втрачено, а поточна неціла точка стає недопустимою в обох гілках. Для вибору змінної розгалуження у програмі застосовано стратегію «найбільш дробової змінної» (most fractional branching): обирається змінна, дробова частина якої найближча до 0,5, тобто з мінімальним значенням

|frac(x□*) − 0,5| → min. (3.5)

Обхід дерева виконується вглиб (depth-first), що мінімізує обсяг пам'яті. Кожен вузол зберігає додаткове обмеження, значення та розв'язок релаксації, статус (розгалуження / цілочисловий / відсічено / недопустимий), що дозволяє побудувати наочну візуалізацію дерева. У реалізації лінійні релаксації розв'язуються методом внутрішніх точок/симплексом бібліотеки SciPy (функція linprog), а структура дерева формується рекурсивно.

Метод потенціалів. Метод потенціалів (метод MODI) - класичний алгоритм оптимізації плану транспортної задачі [2, 3]. Він застосовується до збалансованої задачі (1.13) і складається з побудови початкового опорного плану та його послідовного покращення.

Початковий опорний план у програмі будується методом північно-західного кута: заповнення матриці перевезень починається з верхньої лівої клітинки, у яку записується менше з невикористаного запасу та залишкової потреби

$x_{i}□ = \min(a_i, b□)$, (3.6)

після чого вичерпаний рядок або стовпець виключається з розгляду, і перехід відбувається вниз або праворуч. Коректний опорний план містить рівно m + n − 1 базисну клітинку.

Далі виконується ітераційне покращення плану:

1. обчислюють потенціали постачальників u_i та споживачів v_j із системи рівнянь для всіх базисних клітинок, поклавши $u_1 = 0$:
   $u_i + v_j = c_{ij}$; (3.7)
2. для кожної небазисної клітинки обчислюють оцінку
   $\Delta_{ij} = c_{ij} - u_i - v_j$; (3.8)
3. якщо всі $\Delta_{ij} \geq 0$ - план оптимальний, алгоритм завершується; інакше до базису вводять клітинку з найменшою (найбільш від'ємною) оцінкою;
4. для вхідної клітинки будують замкнений цикл через базисні клітинки, у якому знаки чергуються (+θ, −θ, +θ, …);
5. визначають $\theta = \min x_{ij}$ серед клітинок зі знаком «−»; до клітинок «+» додають θ, від клітинок «−» віднімають θ; клітинка зі знаком «−», що набула нульового значення, виходить із базису;
6. повертаються до кроку 1.

Якщо транспортна задача незбалансована, перед побудовою плану вона автоматично балансується введенням фіктивного постачальника або споживача з нульовими тарифами. Для кожної ітерації програма зберігає матрицю перевезень, потенціали, оцінки та цикл перерозподілу, що дає змогу показати процес оптимізації крок за кроком.

Спільною рисою всіх трьох методів є скінченність: за відсутності виродженості кожен із них досягає оптимуму за скінченну кількість ітерацій. Саме покрокова, детермінована природа цих алгоритмів і покладена в основу навчальної моделі тренажера, описаної далі.

3.2 Проектування архітектури програмного забезпечення

Архітектура програмного забезпечення визначає верхньорівневу структуру системи: набір компонентів, їхні обов'язки, інтерфейси та

принципи взаємодії. Для навчального тренажера обрано **клієнт-серверну архітектуру** з чітким розділенням презентаційного шару (відображення та інтерактивність) і шару обчислень (реалізація алгоритмів). Взаємодія між ними відбувається через REST API у форматі JSON поверх протоколу HTTP.

Обґрунтування вибору клієнт-серверної архітектури.

Альтернативами були монолітний вебзастосунок із серверним рендерингом сторінок та десктопна програма. Клієнт-серверну архітектуру з REST API обрано з таких міркувань: кросплатформність - користувач працює через будь-який сучасний браузер без встановлення додаткового програмного забезпечення; централізація обчислювальної логіки - алгоритми виконуються на сервері, що спрощує їх оновлення та приховує реалізацію від клієнта; незалежність шарів - інтерфейс і алгоритми можна розвивати окремо, а у разі потреби замінити фронтенд (наприклад, на мобільний застосунок), не змінюючи серверну логіку.

Важливою рисою спроєктованої системи є **відсутність стану на сервері (stateless REST)**: кожен HTTP-запит самодостатній і містить усі дані, потрібні для його обробки, а сервер не зберігає інформації про попередні запити. Це спрощує розгортання, масштабування та підвищує надійність. Уся статистика навчання зберігається на боці клієнта у сховищі браузера (localStorage), тож вона персональна для кожного користувача й не потребує серверної бази даних.

Логічно систему поділено на чотири складові, наведені в таблиці 3.1.

Таблиця 3.1 - Складові архітектури тренажера

Складова	Призначення
Клієнт (Frontend)	Односторінковий застосунок (SPA): відображення інтерфейсу, інтерактивні таблиці й діаграми, формування запитів до сервера, зберігання статистики у localStorage

Сервер (Backend)	REST API: приймання й валідація запитів, виклик алгоритмів, формування відповідей у форматі JSON
Обчислювальне ядро	Реалізація алгоритмів (симплекс-метод, метод гілок і меж, метод потенціалів), валідатори дій студента, генератор задач
Сховище даних	Бібліотека готових задач у файлі формату JSON (доступ лише для читання); статистика навчання - у localStorage браузера

Взаємодія компонентів. Клієнт надсилає на сервер запити двох основних типів: запити на повне розв'язання задачі (у відповідь сервер повертає покроковий протокол - послідовність станів алгоритму) та запити на перевірку окремої дії студента (у відповідь сервер повідомляє, правильна дія чи ні, і за потреби надає підказку). Оскільки сервер не зберігає стану, поточний стан розв'язання (наприклад, таблиця, яку перевіряє студент) передається у тілі запиту.

Загальну схему клієнт-серверної архітектури наведено на рис. 3.1.

Рисунок 3.1 - Схема клієнт-серверної архітектури тренажера

Моделювання поведінки системи (UML). Для опису функціональності та взаємодії компонентів використано уніфіковану мову моделювання UML - зокрема діаграму варіантів використання та діаграму послідовності.

Діаграма варіантів використання відображає функції системи з погляду користувача. Основний (і єдиний) актор системи - **студент;** адміністративних ролей у тренажері немає, оскільки бібліотека задач постачається у вигляді файлу, а статистика зберігається локально. Студенту доступні такі варіанти використання: обрати метод (симплекс / гілки й межі / транспортна задача); переглянути теоретичну довідку; обрати задачу з бібліотеки; згенерувати випадкову задачу; ввести власну задачу; розв'язати задачу в одному з трьох режимів (перегляд усіх кроків, покроковий прохід, «сліпий» режим); перевірити свій крок та отримати підказку; переглянути й скинути статистику навчання. Діаграму наведено на рис. 3.2.

Рисунок 3.2 - Діаграма варіантів використання тренажера

Діаграма послідовності відображає динаміку взаємодії об'єктів у часі на прикладі ключового сценарію - перевірки кроку студента під час розв'язання симплекс-методом. Сценарій такий: студент вводить свій вибір (наприклад, ведучий стовпець) у компоненті інтерфейсу; компонент надсилає на сервер HTTP-запит з поточним станом і дією студента; сервер передає дані валідатору обчислювального ядра; валідатор порівнює дію з еталонною та повертає результат із підказкою; сервер формує JSON-відповідь; інтерфейс підсвічує правильність вибору й за потреби показує підказку. Діаграму наведено на рис. 3.3.

Рисунок 3.3 - Діаграма послідовності перевірки кроку симплекс-методу

Структура коду. Відповідно до архітектури, програмний код поділено на дві незалежні частини - серверну та клієнтську. Серверна частина організована за модулями: точка входу та конфігурація вебсервера; модуль маршрутів REST API; обчислювальне ядро (окремі модулі для симплекс-методу, методу гілок і меж, методу потенціалів, валідаторів і генератора

задач); файл бібліотеки задач. Клієнтська частина побудована за компонентним підходом: спільні компоненти (форма введення задачі, вибір задачі з бібліотеки, панель теорії, панель статистики) та три модулі за методами, кожен з яких містить власні компоненти відображення таблиць, дерева чи плану перевезень і покрокового інтерактивного проходу. Детальний опис модулів та їхньої реалізації наведено в розділі 4.

3.3 Обґрунтування вибору програмних засобів реалізації

Технологічний стек обирався з огляду на такі критерії: безкоштовність і відкритість (для вільного використання у навчанні); відповідність обраній клієнт-серверній архітектурі; зрілість і наявність документації; поширеність у сучасній промисловій розробці; придатність до наукових обчислень. За цими критеріями сформовано стек, де серверна частина побудована на мові Python, а клієнтська - на JavaScript-бібліотеці React. Зведену характеристику засобів наведено в таблиці 3.2.

Таблиця 3.2 - Технологічний стек системи

Складова	Засіб	Призначення
Мова сервера	Python	Реалізація алгоритмів і серверної логіки
Вебфреймворк	Flask	Побудова REST API, маршрутизація запитів
Матричні обчислення	NumPy	Зберігання й перетворення симплекс-таблиць, матриць перевезень
Розв'язування ЛП	SciPy (linprog)	Розв'язання лінійних релаксацій у методі гілок і меж
WSGI-сервер	Gunicorn	Запуск серверного застосунку у промисловому середовищі
Мова клієнта	TypeScript	Строга типізація компонентів і даних

UI-бібліотека	React	Побудова інтерактивного інтерфейсу
Компоненти інтерфейсу	Ant Design	Готові елементи UI (форми, таблиці, картки, кнопки)
HTTP-клієнт	axios	Звернення клієнта до REST API
Візуалізація дерева	ReactFlow	Побудова дерева методу гілок і меж
Розгортання клієнта	хмарна платформа (CDN)	Хостинг статичного фронтенду
Розгортання сервера	хмарна платформа	Хостинг серверного застосунку
Контроль версій	Git/GitHub	Зберігання коду, автоматичне розгортання

Серверна частина. Мову **Python** обрано як основну мову серверної частини. Це універсальна високорівнева мова з простим синтаксисом, що є фактичним стандартом у галузі наукових обчислень і дослідження операцій завдяки розвиненій екосистемі бібліотек. Для побудови REST API застосовано мікрофреймворк **Flask** [12]: на відміну від важчих рішень, він мінімалістичний, не нав'язує жорсткої структури проєкту й чудово підходить для невеликого сервісу, основне завдання якого - приймати запит, викликати алгоритм і повертати результат у форматі JSON.

Обчислювальне ядро спирається на дві наукові бібліотеки. **NumPy** [11] використовується для роботи з симплекс-таблицями та матрицями перевезень як з n-вимірними масивами: операції нормування рядка, лінійного виключення та поелементного порівняння виконуються ефективно й лаконічно. **SciPy** [10], а саме функція scipy.optimize.linprog, застосовується для розв'язання лінійних релаксацій у методі гілок і меж - це позбавляє потреби писати власний розв'язувач ЛП для допоміжних підзадач і гарантує чисельну надійність. Власні ж реалізації симплекс-методу та методу

потенціалів побудовано окремо, оскільки тренажеру потрібні не лише кінцеві відповіді, а й повні проміжні таблиці для покрокового показу, чого стандартні бібліотечні функції не надають. Для запуску застосунку у промисловому середовищі використовується WSGI-сервер **Gunicorn**.

Клієнтська частина. Інтерфейс реалізовано на бібліотеці **React** [13] - одній із найпоширеніших технологій побудови вебінтерфейсів. Її компонентний підхід та віртуальний DOM добре пасують до інтерактивних елементів тренажера - симплекс-таблиць із редагованими клітинками, дерева методу гілок і меж, таблиці перевезень. Код клієнта написано мовою **TypeScript**, що додає строгу статичну типізацію та зменшує кількість помилок під час розробки. Для пришвидшення побудови інтерфейсу застосовано бібліотеку готових компонентів **Ant Design** [14] (форми, таблиці, картки, кнопки, повідомлення) з вбудованою українською локалізацією. Звернення до сервера виконуються через HTTP-клієнт **axios**, а для наочної побудови дерева методу гілок і меж використано бібліотеку **ReactFlow**.

Розгортання та супровід. Клієнтську й серверну частини розгорнуто на безкоштовних хмарних платформах: статичний фронтенд розміщено на платформі з глобальною мережею доставки контенту (CDN), а серверний застосунок - на платформі, що запускає Python-процес. Код зберігається в репозиторії **Git** на сервісі **GitHub**, звідки налаштовано автоматичне розгортання: кожне оновлення коду автоматично перебудовує й публікує застосунок. Такий підхід забезпечує доступність тренажера цілодобово без потреби в обслуговуванні власного сервера, що відповідає нефункціональним вимогам, сформульованим у підрозділі 1.1.

Висновок щодо вибору засобів. Обраний стек повністю задовольняє висунуті критерії: усі засоби безкоштовні й відкриті, мають велику спільноту та документацію, підтримують клієнт-серверну архітектуру зі stateless REST API й забезпечують як ефективні обчислення на сервері, так і багатий інтерактивний інтерфейс на клієнті. Поділ на дві незалежні частини дозволяє

розвивати алгоритмічне ядро та інтерфейс окремо, а розгортання на хмарних платформах робить тренажер доступним без додаткових витрат.

4. ПРАКТИЧНА ЧАСТИНА

4.1 Опис програмної реалізації тренажера

Реалізацію виконано відповідно до спроєктованої у розділі 3 клієнт-серверної архітектури. Код поділено на дві незалежні частини - серверну (Python/Flask) та клієнтську (React/TypeScript), які взаємодіють через REST API. Розробку вели з використанням системи контролю версій Git; обидві частини розгорнуто на хмарних платформах з автоматичним оновленням після кожної зміни коду.

Структура проєкту. Серверну частину організовано за принципом поділу відповідальності: точка входу та конфігурація вебсервера винесені окремо від маршрутів API, а ті - окремо від обчислювального ядра. Узагальнену структуру серверної частини наведено нижче:

- модуль точки входу - створення та конфігурування застосунку Flask, налаштування механізму крос-доменних запитів (CORS), реєстрація маршрутів; - модуль маршрутів - оголошення всіх ендпоінтів REST API; - обчислювальне ядро - окремі модулі: розв'язувач симплекс-методу, розв'язувач методу гілок і меж, розв'язувач транспортної задачі, валідатори дій студента, генератор задач; - файл бібліотеки задач у форматі JSON.

Клієнтську частину побудовано за компонентним підходом: спільні компоненти (форма введення задачі, вибір задачі з бібліотеки, панель теорії, панель статистики), глобальний стан застосунку (зберігання статистики у localStorage), типізований клієнт API та три модулі за методами - симплекс-метод, метод гілок і меж, транспортна задача, кожен зі своїми компонентами відображення й покрокового проходу.

Точка входу сервера. Серверний застосунок створюється за шаблоном «фабрика застосунку» (application factory): окрема функція конфігурує екземпляр Flask, що дозволяє створювати незалежні екземпляри, зокрема для тестування. Механізм CORS налаштовується через змінну середовища: у промисловому режимі дозволяються запити лише з домену клієнта, локально - з будь-якого джерела. Порт також береться зі змінної середовища, яку надає хмарна платформа.

Обчислювальне ядро. Ядро містить три розв'язувачі та допоміжні модулі. Спільний принцип усіх розв'язувачів - не лише отримати кінцеву відповідь, а й зберегти повний протокол розв'язання (послідовність станів алгоритму), який потім передається клієнту для покрокового відображення.

Розв'язувач симплекс-методу реалізує табличний симплекс-метод. Він зводить задачу до канонічної форми (додає балансові змінні), будує початкову таблицю та виконує ітерації, на кожній з яких зберігає «знімок» таблиці з позначеними ведучими рядком і стовпцем. Вибір ведучого стовпця виконується за правилом Данціга, ведучого рядка - за правилом мінімального відношення (3.3), перетворення таблиці - Гаусовим виключенням. Фрагмент логіки однієї ітерації наведено в лістингу 4.1.

Лістинг 4.1 - Вибір ведучого елемента та умова необмеженості (фрагмент)

```
# ведучий стовпець — за правилом Данціга
deltas = tableau[-1, :n_total]
pivot_col = int(np.argmin(deltas))      # для внутрішньої мінімізації

# ведучий рядок — за правилом мінімального відношення
col = tableau[:-1, pivot_col]
rhs = tableau[:-1, -1]
ratios = np.where(col > 1e-9, rhs / col, np.inf)
if np.all(np.isinf(ratios)):
```

```
        return {"status": "unbounded"}      # задача необмежена
    pivot_row = int(np.argmin(ratios))
```

Розв'язувач методу гілок і меж реалізує рекурсивний перебір із відсіканням. На кожному вузлі розв'язується лінійна релаксація за допомогою функції scipy.optimize.linprog; далі вузол або відсікається (недопустимий, гірший за рекорд), або дає цілий розв'язок (оновлює рекорд), або розгалужується. Змінна розгалуження обирається за стратегією «найбільш дробової» (3.5), після чого створюються дві підзадачі з додатковими обмеженнями (3.4). Кожен вузол зберігається з ідентифікатором, посиланням на батьківський вузол, додатковим обмеженням, значенням і розв'язком релаксації та статусом, що дозволяє клієнту побудувати дерево.

Розв'язувач транспортної задачі будує початковий опорний план методом північно-західного кута, за потреби попередньо балансуючи задачу фіктивним постачальником або споживачем, а потім оптимізує його методом потенціалів: обчислює потенціали (3.7), оцінки небазисних клітинок (3.8), за наявності від'ємної оцінки будує замкнений цикл і виконує перерозподіл. Кожна ітерація зберігається з матрицею перевезень, потенціалами, оцінками та циклом.

Валідатори - це набір функцій, що перевіряють дії студента в режимі покрокового навчання. Вони порівнюють введену студентом дію з еталонною (обчисленою розв'язувачем) і повертають уніфіковану структуру: ознаку правильності, пояснювальне повідомлення, правильну відповідь та підказку. Реалізовано перевірки вибору ведучого стовпця та рядка, перерахунку таблиці у «сліпому» режимі (з вказівкою конкретних помилкових клітинок), вибору змінної розгалуження, побудови плану перевезень, обчислення потенціалів та вибору вхідної клітинки.

Генератор задач створює випадкові задачі заданої розмірності у гарантовано коректному діапазоні (зокрема з невід'ємними правими частинами обмежень), що дозволяє студенту тренуватися на необмеженій кількості нових прикладів.

REST API. Усі маршрути згруповано під спільним префіксом /api. Вони поділяються на інформаційні, маршрути бібліотеки задач, генератора, розв'язувачів (повертають покрокові розв'язки) та валідаторів (перевіряють дії студента). Запити й відповіді передаються у форматі JSON; вхідні дані суворо перевіряються на наявність обов'язкових полів і типи, а у разі помилки повертається структурована відповідь з повідомленням. Основні групи маршрутів наведено в таблиці 4.1.

Таблиця 4.1 - Основні групи маршрутів REST API

Метод	Маршрут	Призначення
GET	/api/tasks?type=…	Список задач певного типу з бібліотеки
GET	/api/tasks/random?type=…	Випадкова задача з бібліотеки
GET	/api/generate?type=…&vars=…&constraints=…	Згенерувати нову задачу
POST	/api/simplex	Покроковий розв'язок симплекс-методом
POST	/api/branch-and-bound	Дерево розв'язку методом гілок і меж
POST	/api/transport	Ітерації розв'язку транспортної задачі
POST	/api/simplex/check	Перевірка вибору ведучого стовпця/рядка
POST	/api/simplex/check-tableau	Перевірка перерахунку таблиці («сліпий» режим)
POST	/api/branch-and-bound/check	Перевірка вибору змінної

		розгалуження
POST	/api/transport/check-potentials	Перевірка обчислених потенціалів
POST	/api/transport/check-entering	Перевірка вибору вхідної клітинки

Клієнтська частина. Інтерфейс - односторінковий застосунок із верхньою панеллю для перемикання між методами та маршрутизацією на боці клієнта. Звернення до сервера інкапсульовано в типізованому модулі-клієнті над axios, що повертає строго типізовані результати. Глобальний стан (історія спроб, помилок і підказок) зберігається через механізм контексту React і дублюється в localStorage браузера, тож статистика персональна та зберігається між сесіями.

Кожен із трьох методів подано окремим модулем зі схожою будовою: головний компонент із формою введення задачі та перемикачем режиму відображення; компонент таблиці (симплекс-таблиці, таблиці перевезень) або дерева (для методу гілок і меж); компонент покрокового інтерактивного проходу, що містить квіз для вибору дії, перевірку через відповідний маршрут API та блоки з поясненнями. Особливістю реалізації є компоненти деталізованих обчислень, які показують саме арифметику кроку: для симплекс-методу - нормування ведучого рядка та виключення за Гаусом із підстановкою чисел у кожну клітинку; для транспортної задачі - перерозподіл за циклом із позначенням клітинок, що набувають або втрачають вантаж. Завдяки цьому студент бачить не лише, *що* зробити, а й *як* саме перерахувати таблицю.

4.2 Перевірка роботи програми та інструкція користувача

Перевірка коректності розв'язувачів. Коректність обчислювального ядра перевірено двома способами - модульним тестуванням і незалежною звіркою результатів.

Для серверної частини створено набір модульних тестів (з використанням бібліотеки pytest), які перевіряють: правильність розв'язків базових задач симплекс-методом, зокрема граничні випадки (необмеженість,

виродженість); відповідність результатів методу гілок і меж відомим цілочисловим оптимумам; коректність балансування, побудови початкового плану та циклів у транспортній задачі; роботу валідаторів на правильних і хибних відповідях; коректність HTTP-рівня (статуси та формат відповідей API).

Незалежну звірку виконано зіставленням результатів власних розв'язувачів із результатами бібліотеки SciPy (функції scipy.optimize.linprog та scipy.optimize.milp): на серії контрольних задач оптимальні значення цільової функції збігаються з незалежним розв'язувачем у межах похибки $1 \cdot 10^{-5}$. Додатково перевірено консистентність покрокових переходів: кожна симплекс-таблиця отримується з попередньої коректним Гаусовим виключенням, а кожен цикл перерозподілу транспортної задачі зберігає баланс запасів і потреб.

Контрольний приклад. Розглянемо роботу тренажера на прикладі задачі лінійного програмування з підрозділу 1.2 (формули (1.5)–(1.7)):

$Z = 3x_1 + 4x_2 \rightarrow \max, x_1 + 2x_2 \leq 14, 3x_1 + 2x_2 \leq 18, x_1, x_2 \geq 0.$

Симплекс-метод розв'язує її за дві ітерації. Хід розв'язання подано в таблиці 4.2

Таблиця 4.2 - Хід розв'язання контрольного прикладу симплекс-методом

Ітерація	Базис	Ведучий стовпець	Ведучий рядок	Значення Z
0 (початкова)	s_1, s_2	x_2	s_1 ($\theta = 7$)	0
1	x_2, s_2	x_1	s_2 ($\theta = 4$)	28
2 (оптимум)	x_2, x_1	-	-	34

Оптимальний розв'язок: $x_1 = 4$, $x_2 = 5$, $Z = 34$. Той самий результат дає незалежна перевірка бібліотекою SciPy, що підтверджує коректність розв'язувача. Аналогічно перевірено роботу методу гілок і меж (на

цілочисловій версії задачі) та методу потенціалів (на типовій транспортній задачі 3×3).

Інструкція користувача. Робота з тренажером не потребує встановлення - достатньо відкрити вебсторінку застосунку в браузері. Інтерфейс українськомовний; перемикання між методами відбувається через верхню панель.

Загальний порядок роботи такий:

1. обрати метод на верхній панелі - симплекс-метод, метод гілок і меж або транспортна задача;
2. за потреби розгорнути панель теорії, щоб ознайомитися з математичною моделлю та алгоритмом обраного методу;
3. задати умову задачі одним зі способів: обрати готову задачу з бібліотеки, згенерувати випадкову задачу заданої розмірності або ввести власну у формі;
4. обрати режим роботи та натиснути «Розв'язати».

Головний екран із формою введення задачі та панеллю теорії показано на рис. 4.1.

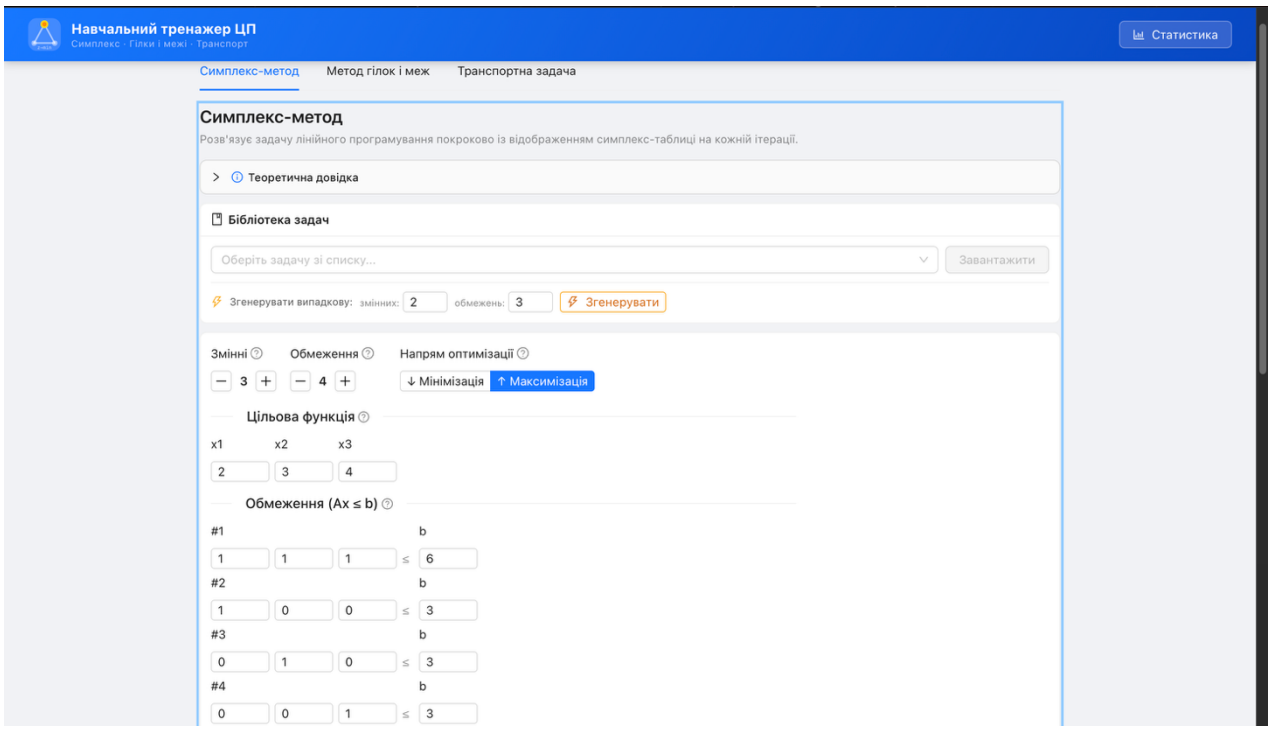


Рисунок 4.1 - Головний екран тренажера з формою введення задачі

Тренажер підтримує чотири режими роботи, що відповідають зростанню складності:

- **режим «всі кроки»** - оглядовий перегляд готового розв'язку: усі симплекс-таблиці (вузли дерева, ітерації) показано послідовно для першого ознайомлення; - **режим «крок за кроком»** - інтерактивний прохід, де на кожній ітерації студент сам обирає дію (ведучий стовпець і рядок, змінну розгалуження, вхідну клітинку), отримує перевірку та пояснення, а також може розгорнути блок детальних обчислень; - **«сліпий» режим** (для симплекс-методу) - найскладніший: студент самостійно перераховує наступну таблицю, а програма вказує конкретні помилкові клітинки.

Окремо реалізовано **«тестовий» режим,** призначений для самоконтролю та оцінювання набутих навичок. На відміну від навчальних режимів, він не показує готового розв'язку: тренажер генерує випадкову задачу, а студент послідовно відповідає на серію питань за її ітераціями — обирає ведучий стовпець і рядок, обчислює потенціали та оцінки клітин тощо. Передбачено два підрежими: тренувальний — із миттєвою перевіркою кожної відповіді, поясненням і необмеженою кількістю спроб, та

контрольний — із однією спробою на питання й перевіркою лише наприкінці. За результатами проходження виводиться підсумковий бал, що дозволяє студентові об'єктивно оцінити рівень засвоєння методу, а викладачеві — використовувати тренажер як інструмент поточного контролю. Правильність відповідей перевіряється тими самими валідаторами, що й у покроковому режимі, тож еталоном слугує розв'язок, обчислений власним розв'язувачем.

Приклад інтерактивного покрокового проходу із вибором ведучого елемента та перевіркою показано на рис. 4.2, а блок детальних обчислень pivot-операції - на рис. 4.3. Приклад проходження тесту в тренувальному підрежимі - з питанням, миттєвою перевіркою та поясненням - показано на рис. 4.4.

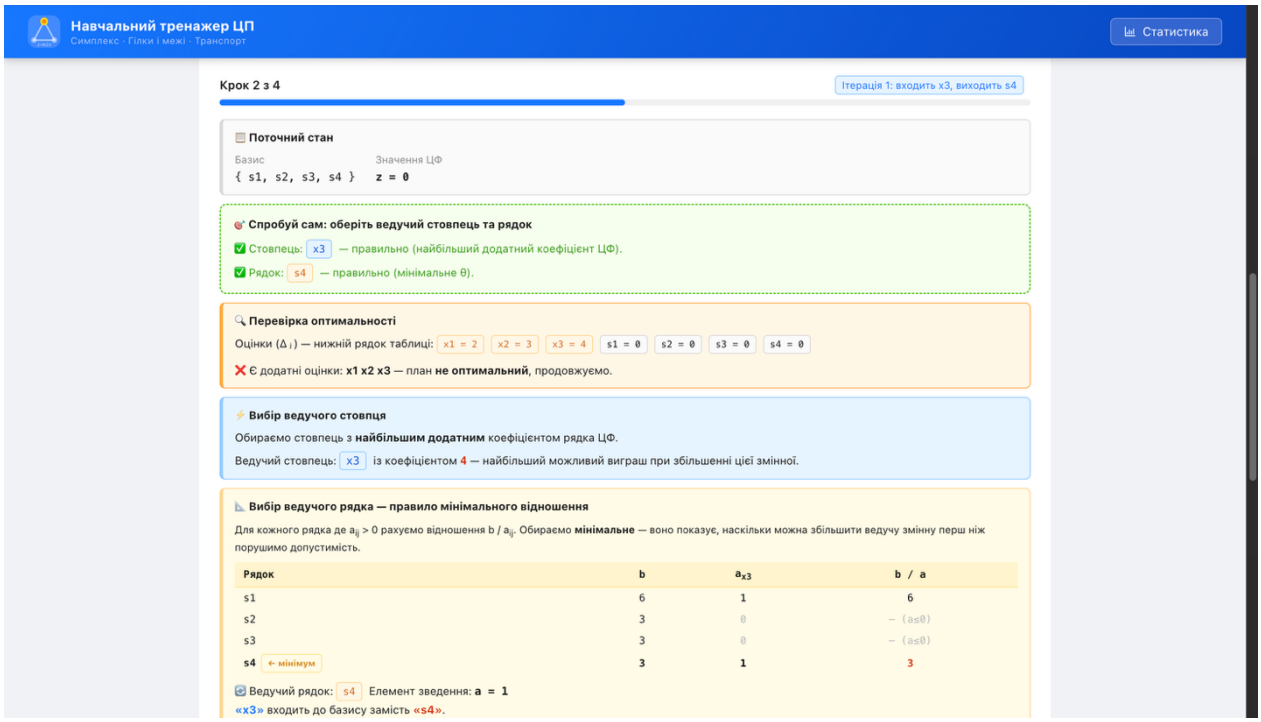


Рисунок 4.2 - Інтерактивний покроковий прохід симплекс-методу

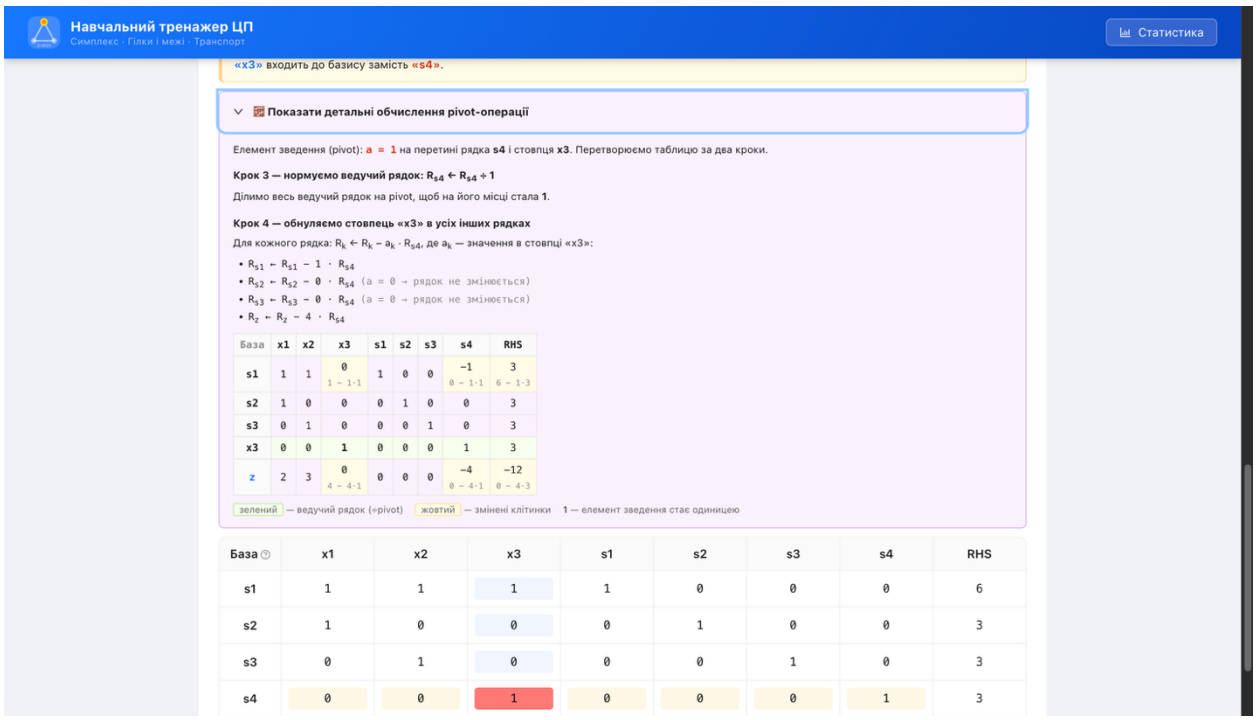


Рисунок 4.3 - Детальні обчислення pivot-операції

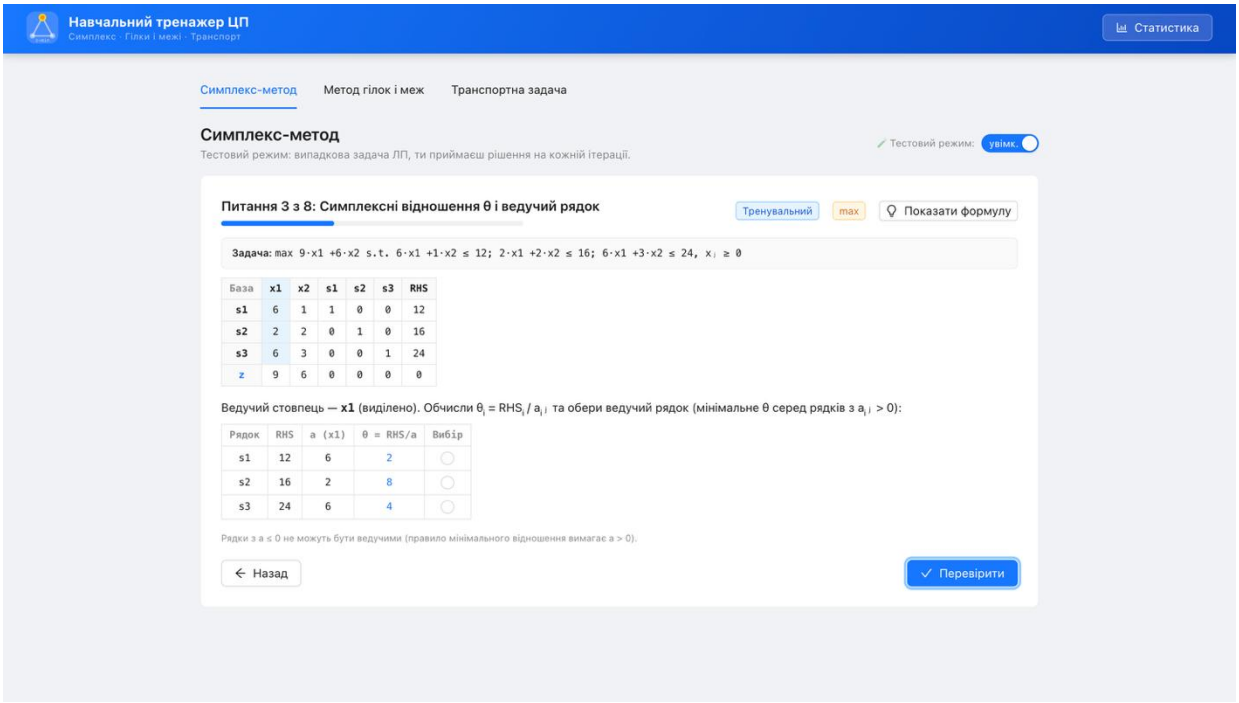


Рисунок 4.4 - Інтерфейс тестового режиму тренажера

Для методу гілок і меж результат показується у вигляді дерева вузлів із зазначенням обмежень, значень релаксації та статусу кожного вузла (розгалуження, цілочисловий, відсічено, недопустимий). Для транспортної задачі на кожній ітерації відображаються матриця перевезень, потенціали, оцінки небазисних клітинок та цикл перерозподілу. Візуалізацію дерева

методу гілок і меж наведено на рис. 4.5, а крок розв'язання транспортної задачі - на рис. 4.6.

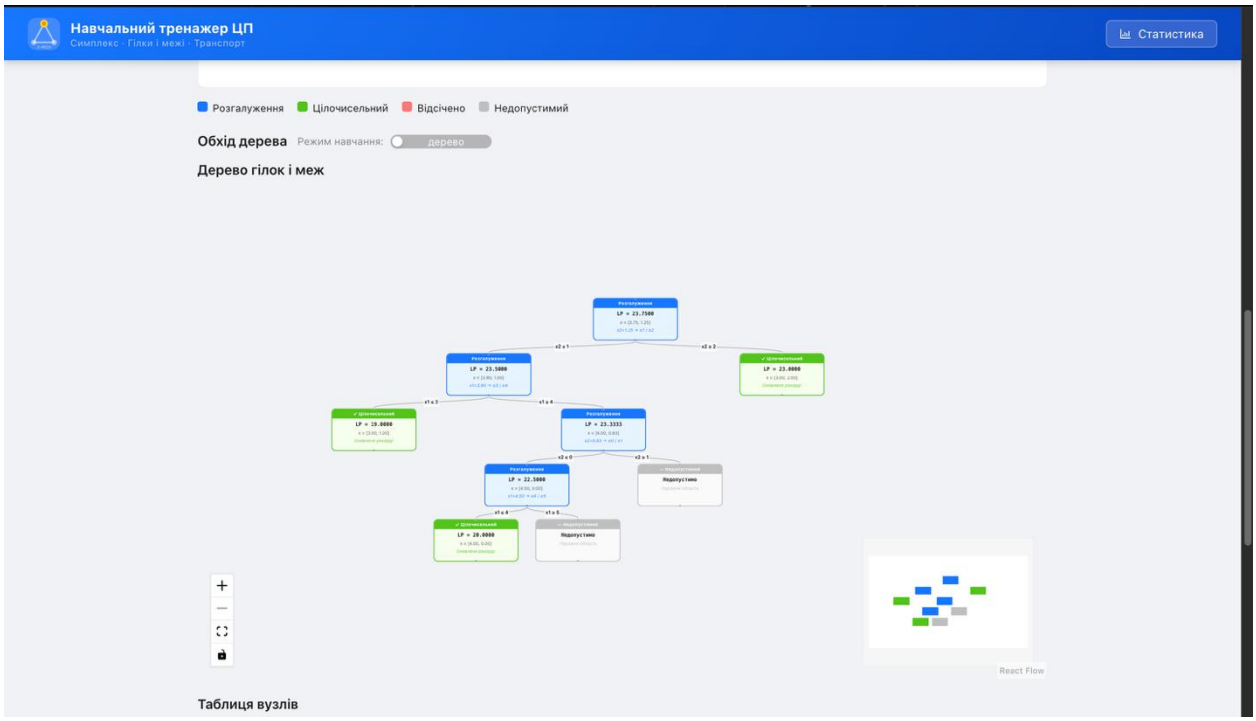


Рисунок 4.5 - Візуалізація дерева методу гілок і меж

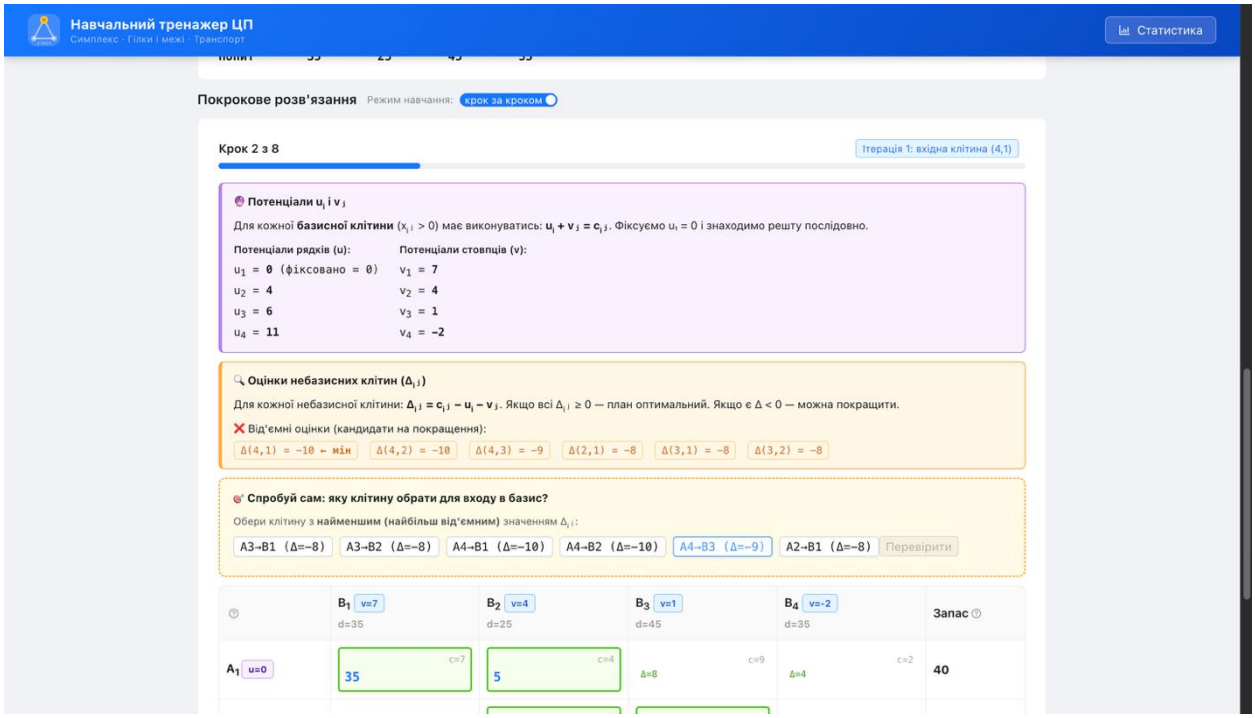


Рисунок 4.6 - Крок розв'язання транспортної задачі методом потенціалів

Під час роботи тренажер веде статистику навчання - кількість спроб, помилок і використаних підказок, - яку студент може переглянути на панелі статистики та за потреби скинути. Статистика зберігається локально у браузері, тож є персональною та доступною між сесіями.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи спроєктовано та програмно реалізовано навчальний вебтренажер для опанування методів розв'язування задач лінійного та цілочислового програмування у межах курсу «Методи оптимізації та дослідження операцій». Отримано такі основні результати.

1. Проаналізовано предметну область - задачі лінійного та цілочислового програмування - і сформульовано постановку задачі розробки. Побудовано математичні моделі задачі лінійного (цілочислового) програмування та транспортної задачі, визначено їхні характеристики й відмінності, що зумовлюють вибір методів розв'язання. На основі аналізу сформульовано десять функціональних і низку нефункціональних вимог до тренажера.
2. Оглянуто наявні програмні рішення - промислові розв'язувачі (CPLEX, Gurobi, LINDO, MATLAB Optimization Toolbox), відкриті бібліотеки (PuLP, SciPy, OR-Tools) та навчальні онлайн-сервіси. За критеріями придатності для навчання виявлено, що жодне з них одночасно не забезпечує покрокової візуалізації кількох методів, перевірки дій студента та українськомовного інтерфейсу. Це обґрунтувало доцільність власної розробки.
3. Досліджено три класичні методи, що становлять алгоритмічне ядро тренажера: симплекс-метод, метод гілок і меж та метод потенціалів. Для кожного описано принцип роботи, послідовність кроків та умову оптимальності у тому вигляді, в якому їх реалізовано в програмі.
4. Спроєктовано клієнт-серверну архітектуру системи зі stateless REST API, що чітко розділяє презентаційний шар і шар обчислень.

Побудовано UML-діаграми (варіантів використання та послідовності) та схему архітектури, обґрунтовано вибір технологічного стека (Python, Flask, NumPy, SciPy на сервері; React, TypeScript, Ant Design на клієнті).

5. Реалізовано програмне забезпечення тренажера: серверне обчислювальне ядро з власними розв'язувачами трьох методів, валідаторами дій студента та генератором задач; REST API; клієнтський односторінковий застосунок із трирівневою прогресією складності навчання (перегляд готового розв'язку, інтерактивний покроковий прохід із валідацією, «сліпий» режим) та деталізованим показом обчислень кожного кроку. Тренажер розгорнуто на хмарних платформах із цілодобовим доступом.
6. Перевірено коректність роботи тренажера модульним тестуванням і незалежною звіркою результатів із бібліотекою SciPy: оптимальні значення збігаються з еталонними у межах похибки $1 \cdot 10^{-5}$. Підготовлено інструкцію користувача та продемонстровано роботу системи на контрольних прикладах для всіх трьох методів.

Поставлену мету досягнуто: створено працездатний навчальний інструмент, який, на відміну від наявних аналогів, поєднує в єдиному українськомовному вебсередовищі теорію, бібліотеку й генератор задач, покрокову візуалізацію та перевірку дій студента для трьох методів цілочислового програмування.

РЕКОМЕНДАЦІЇ

За результатами роботи можна сформулювати такі рекомендації.

Розроблений тренажер рекомендовано до використання в освітньому процесі кафедри комп'ютерних наук та інформаційних технологій: у дистанційному курсі «Методи оптимізації та дослідження операцій» для самостійної роботи студентів, як інструмент проведення лабораторних і практичних занять та як засіб самоконтролю перед контрольними роботами. Бібліотека готових задач і генератор випадкових задач дозволяють викладачеві формувати індивідуальні завдання й уникати «вивчення відповідей напам'ять».

Подальший розвиток тренажера доцільно спрямувати на: розширення переліку методів (двоїстий симплекс-метод і двофазний метод штучного базису для задач з обмеженнями «≥» та «=», метод відсікаючих площин Гоморі); запровадження правила Бленда та коректної обробки виродженості для підвищення стійкості алгоритмів; додавання експорту покрокового протоколу розв'язання у формат PDF; інтеграцію з системами дистанційного навчання; розширення модуля статистики для виявлення типових утруднень студентів.

СПИСОК ЛІТЕРАТУРИ

1. Наконечний С. І., Савіна С. С. Математичне програмування : навч. посібник. Київ : КНЕУ, 2003. 452 с. URL: https://kneu.edu.ua/ua/science_kneu/scientific_schools/mtrve/mtrve_praci/mtrve_prazi/matprognpos/ (дата звернення: 20.05.2026).
2. Нефьодов Ю. М., Балицька Т. Ю. Методи оптимізації в прикладах і задачах : навч. посібник. Київ : Кондор, 2011. 324 с. URL: https://ir.nmu.org.ua/server/api/core/bitstreams/60240f1a-fb90-40ff-8ba8-471497e105c7/content (дата звернення: 20.05.2026).
3. Зайченко Ю. П. Дослідження операцій : підручник. 8-ме вид., доп. Київ : Слово, 2021. 816 с.
4. Hillier F. S., Lieberman G. J. Introduction to Operations Research. 11th ed. New York : McGraw-Hill Education, 2021. 1056 p.
5. Wolsey L. A. Integer Programming. 2nd ed. Hoboken, NJ : John Wiley & Sons, 2020. 336 p.
6. Dantzig G. B. Linear Programming and Extensions. Princeton, NJ : Princeton University Press, 2016. 656 p.
7. IBM ILOG CPLEX Optimization Studio. Documentation. URL: https://www.ibm.com/docs/en/icos (дата звернення: 18.05.2026).
8. Gurobi Optimizer Reference Manual. Gurobi Optimization, LLC, 2024. URL: https://www.gurobi.com/documentation/ (дата звернення: 18.05.2026).
9. PuLP: A Linear Programming Toolkit for Python. URL: https://github.com/coin-or/pulp (дата звернення: 18.05.2026).

10. SciPy documentation: optimization (linprog, milp). URL: https://docs.scipy.org/doc/scipy/reference/optimize.html (дата звернення: 18.05.2026).
11. NumPy documentation. URL: https://numpy.org/doc/stable/ (дата звернення: 18.05.2026).
12. Flask Documentation. Pallets Projects. URL: https://flask.palletsprojects.com/ (дата звернення: 18.05.2026).
13. React Documentation. Meta Open Source. URL: https://react.dev/ (дата звернення: 18.05.2026).
14. Ant Design - A UI Design Language. URL: https://ant.design/ (дата звернення: 18.05.2026).
15. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.

Додаток А

Програмний код серверної частини

У цьому додатку наведено код ключових модулів серверної частини навчального тренажера: точки входу Flask-додатку, трьох розв'язувачів (симплекс-метод, метод гілок і меж, метод потенціалів), а також репрезентативні фрагменти валідаторів дій студента, генератора задач і REST API. Усі модулі написано на Python 3.12 із використанням бібліотек NumPy 1.26.4 та SciPy 1.13.1; коментарі та повідомлення про помилки — українською мовою.

Лістинг А.1 — backend/app.py

Реалізує шаблон application factory: функція create_app() створює й конфігурує екземпляр Flask, налаштовує CORS (з обмеженням домену через змінну середовища FRONTEND_URL) та реєструє Blueprint з REST-маршрутами.

```
import os
from flask import Flask
from flask_cors import CORS

from api.routes import api_bp



def create_app() -> Flask:
    """Application factory — creates and configures the Flask app."""
    app = Flask(__name__)
    frontend_url = os.environ.get("FRONTEND_URL")
    if frontend_url:
        CORS(app, resources={r"/api/*": {"origins": frontend_url}})
    else:
        CORS(app)
    app.register_blueprint(api_bp, url_prefix="/api")
    return app



# Exposed at module level so Gunicorn can find it: gunicorn app:application
application = create_app()



if __name__ == "__main__":
    debug_mode = os.environ.get("FLASK_DEBUG", "0") == "1"
    port = int(os.environ.get("PORT", 5001))
    application.run(debug=debug_mode, host="0.0.0.0", port=port)
```

Лістинг А.2 — backend/core/simplex_solver.py

Табличний симплекс-метод для задач у формі min/max c·x за умов Ax ≤ b, b ≥ 0, x ≥ 0. Вибір ведучого стовпця — за правилом Данціга, ведучого рядка — за правилом мінімального відношення. Кожна ітерація зберігається у списку steps для покрокового відображення.

```
import numpy as np
from typing import List, Dict, Any, Optional



def solve_simplex(c, A, b, maximize=False):
    """Solve LP: min/max c^T x s.t. Ax <= b, x >= 0. Returns step-by-step tableaux."""
    c_arr = np.array(c, dtype=float)
    A_arr = np.array(A, dtype=float)
    b_arr = np.array(b, dtype=float)


    if np.any(b_arr < -1e-9):
        return {"error": "Всі значення правої частини b мають бути >= 0"}


    if maximize:
        c_arr = -c_arr


    m, n = A_arr.shape
    n_total = n + m  # original vars + slack vars


    # Build tableau: m constraint rows + 1 objective row; cols [x | s | RHS]
    T = np.zeros((m + 1, n_total + 1))
    T[:m, :n] = A_arr
    T[:m, n:n_total] = np.eye(m)
    T[:m, -1] = b_arr
    T[m, :n] = c_arr


    basic = list(range(n, n_total))  # initial basis: slack variables
    col_names = [f"x{i+1}" for i in range(n)] + [f"s{i+1}" for i in range(m)]
    steps: List[Dict] = []


    def snapshot(desc, p_row=None, p_col=None):
        tableau = [row[:] for row in T.tolist()]
        if maximize:
            tableau[m][:-1] = [-v for v in tableau[m][:-1]]
            tableau[m][-1] = -tableau[m][-1]
        steps.append({
            "description": desc, "tableau": tableau,
            "col_names": col_names + ["RHS"],
            "row_names": [col_names[v] for v in basic] + ["z"],
            "basic_vars": [col_names[v] for v in basic],
            "pivot_row": p_row, "pivot_col": p_col,
        })


    snapshot("Початкова таблиця")


    num_pivots = 0
    for iteration in range(200):
        obj_row = T[m, :n_total]
        if np.all(obj_row >= -1e-9):       # optimality: all reduced costs >= 0
            break
        p_col = int(np.argmin(obj_row))    # entering: Dantzig's rule
        col_vals = T[:m, p_col]
        if np.all(col_vals <= 1e-9):
            return {"error": "Задача необмежена", "steps": steps}
        rhs = T[:m, -1]                    # leaving: minimum ratio test
        with np.errstate(divide="ignore", invalid="ignore"):
            ratios = np.where(col_vals > 1e-9, rhs / col_vals, np.inf)
        p_row = int(np.argmin(ratios))
```

```
        entering = col_names[p_col]
        leaving = col_names[basic[p_row]]
        snapshot(f"Ітерація {iteration+1}: входить {entering}, виходить {leaving}", p_row,
p_col)

        T[p_row] = T[p_row] / T[p_row, p_col]   # pivot: normalize + eliminate
        for i in range(m + 1):
            if i != p_row:
                T[i] -= T[i, p_col] * T[p_row]
        basic[p_row] = p_col
        num_pivots += 1

    x = np.zeros(n_total)
    for i, var in enumerate(basic):
        x[var] = T[i, -1]
    solution = [round(float(v), 10) for v in x[:n]]
    obj_value = float(T[m, -1]) if maximize else -float(T[m, -1])

    if num_pivots == 0:
        steps[0]["description"] = "Оптимальна таблиця (початкове рішення вже оптимальне)"
    else:
        snapshot("Оптимальна таблиця")

    return {"status": "optimal", "solution": solution,
            "optimal_value": round(obj_value, 10),
            "steps": steps, "num_iterations": num_pivots}
```

Лістинг А.3 — backend/core/bnb_solver.py

Метод гілок і меж для цілочислових задач. LP-релаксацію на кожному вузлі розв'язує scipy.optimize.linprog (метод HiGHS). Стратегія розгалуження — most fractional (найближча до 0,5 змінна); відсікання — за правилом меж.

```
import math
import numpy as np
from scipy.optimize import linprog
from typing import List, Dict, Any, Optional, Tuple



def solve_branch_and_bound(c, A_ub, b_ub, maximize=False, var_bounds=None):
    """Solve ILP: min/max c^T x s.t. A_ub x <= b_ub, x >= 0, x integer."""
    c_arr = np.array(c, dtype=float)
    A_arr = np.array(A_ub, dtype=float)
    b_arr = np.array(b_ub, dtype=float)
    n = len(c)
    c_min = -c_arr if maximize else c_arr      # internally always minimize

    if var_bounds is None:
        bounds = [(0.0, None)] * n
    else:
        bounds = [(float(b[0]), float(b[1]) if b[1] is not None else None) for b in
var_bounds]

    nodes: List[Dict[str, Any]] = []
    _id = [0]; _best_obj = [math.inf]; _best_sol = [None]

    def _new_id():
        nid = _id[0]; _id[0] += 1; return nid

    def _solve_lp(nb):
        return linprog(c_min, A_ub=A_arr, b_ub=b_arr, bounds=nb, method="highs")
```

```
def _bnb(parent_id, node_bounds, depth, label):
    nid = _new_id()
    result = _solve_lp(node_bounds)
    node = {"id": nid, "parent_id": parent_id, "depth": depth,
            "label": label, "bounds": [[b[0], b[1]] for b in node_bounds]}


    if result.status != 0:                # infeasible subproblem
        node.update({"status": "infeasible", "lp_value": None, "solution": None})
        nodes.append(node); return


    lp_min = float(result.fun)
    lp_display = -lp_min if maximize else lp_min
    node["lp_value"] = round(lp_display, 8)
    node["solution"] = [round(float(v), 8) for v in result.x]


    if lp_min >= _best_obj[0] - 1e-9:       # prune by bound
        node["status"] = "pruned"; nodes.append(node); return


    frac_idx = None; frac_max = 0.0         # most fractional variable
    for i, val in enumerate(result.x):
        frac_part = val - math.floor(val)
        dist = min(frac_part, 1.0 - frac_part)
        if dist > 1e-6 and dist > frac_max:
            frac_max = dist; frac_idx = i


    if frac_idx is None:                    # all integer -> candidate
        node["status"] = "integer"
        if lp_min < _best_obj[0]:
            _best_obj[0] = lp_min; _best_sol[0] = result.x.copy()
        nodes.append(node); return


    node["status"] = "branched"             # branch on frac_idx
    node["branch_var"] = frac_idx
    node["branch_value"] = round(float(result.x[frac_idx]), 8)
    nodes.append(node)


    frac_val = result.x[frac_idx]
    floor_val = math.floor(frac_val); ceil_val = math.ceil(frac_val)
    var_label = f"x{frac_idx + 1}"
    lb, ub = node_bounds[frac_idx]


    left = list(node_bounds); left[frac_idx] = (lb, float(floor_val))
    _bnb(nid, left, depth + 1, f"{var_label} <= {floor_val}")
    right = list(node_bounds); right[frac_idx] = (float(ceil_val), ub)
    _bnb(nid, right, depth + 1, f"{var_label} >= {ceil_val}")


_bnb(None, bounds, 0, "LP-релаксація")


if _best_sol[0] is None:
    return {"status": "infeasible", "nodes": nodes, "total_nodes": len(nodes)}


raw_obj = float(_best_obj[0])
optimal_value = -raw_obj if maximize else raw_obj
return {"status": "optimal",
        "solution": [round(float(v), 8) for v in _best_sol[0]],
        "optimal_value": round(optimal_value, 8),
        "nodes": nodes, "total_nodes": len(nodes)}
```

Лістинг А.4 — backend/core/transport_solver.py

Метод потенціалів (MODI) для збалансованої транспортної задачі: балансування фіктивним учасником, початковий план методом північно-західного кута, обчислення потенціалів і оцінок, пошук циклу перерозподілу

через DFS та перерозподіл. Нижче — ядро розв'язувача (допоміжні функції _compute_potentials і _find_loop наведено повністю).

```
import numpy as np
from typing import List, Dict, Any, Optional, Tuple, Set



def solve_transport(supply, demand, costs):
    """Solve the balanced transportation problem via the Method of Potentials (MODI)."""
    supply = [float(s) for s in supply]
    demand = [float(d) for d in demand]
    costs_arr = np.array(costs, dtype=float)
    m_orig, n_orig = len(supply), len(demand)


    total_supply, total_demand = sum(supply), sum(demand)
    dummy_col = dummy_row = False               # balance the problem
    if total_supply > total_demand + 1e-9:
        demand += [total_supply - total_demand]
        costs_arr = np.hstack([costs_arr, np.zeros((m_orig, 1))]); dummy_col = True
    elif total_demand > total_supply + 1e-9:
        supply += [total_demand - total_supply]
        costs_arr = np.vstack([costs_arr, np.zeros((1, n_orig))]); dummy_row = True


    m, n = len(supply), len(demand)


    # North-West corner initial basic feasible solution
    allocation = np.zeros((m, n)); basic_cells = []
    sup, dem = supply.copy(), demand.copy(); i = j = 0
    while i < m and j < n:
        amt = min(sup[i], dem[j]); allocation[i, j] = amt
        sup[i] -= amt; dem[j] -= amt
        if (i, j) not in basic_cells: basic_cells.append((i, j))
        if sup[i] < 1e-10 and dem[j] < 1e-10:
            if i + 1 < m: i += 1
            else: j += 1
        elif sup[i] < 1e-10: i += 1
        else: j += 1


    steps = []


    def _compute_potentials():
        u = [None]*m; v = [None]*n; u[0] = 0.0; changed = True
        while changed:
            changed = False
            for (bi, bj) in basic_cells:
                if u[bi] is not None and v[bj] is None:
                    v[bj] = costs_arr[bi, bj] - u[bi]; changed = True
                elif v[bj] is not None and u[bi] is None:
                    u[bi] = costs_arr[bi, bj] - v[bj]; changed = True
        return u, v


    def _find_loop(enter_r, enter_c):           # improvement loop via DFS
        bset = set(basic_cells)
        def dfs(path, move_col):
            r, c = path[-1]; path_set = set(path[1:])
            if move_col:
                for (br, bc) in bset:
                    if bc == c and br != r and (br, bc) not in path_set:
                        res = dfs(path + [(br, bc)], False)
                        if res: return res
            else:
                for (br, bc) in bset:
                    if br == r and bc != c and (br, bc) not in path_set:
                        if bc == enter_c and len(path) >= 3:
                            return path + [(br, bc)]
                        res = dfs(path + [(br, bc)], True)
                        if res: return res
            return None
```

```
        return dfs([(enter_r, enter_c)], False)

    num_iterations = 0
    for iteration in range(200):
        u, v = _compute_potentials()
        bset = set(basic_cells); entering = None; min_delta = -1e-9
        for ii in range(m):
            for jj in range(n):
                if (ii, jj) not in bset and u[ii] is not None and v[jj] is not None:
                    d = costs_arr[ii, jj] - u[ii] - v[jj]
                    if d < min_delta: min_delta = d; entering = (ii, jj)
        if entering is None:                    # optimality reached
            break
        loop = _find_loop(entering[0], entering[1])
        if loop is None:
            return {"error": "Не вдалося знайти цикл покращення (вироджена задача)"}
        minus_cells = [loop[k] for k in range(1, len(loop), 2)]
        theta = min(allocation[r, c] for r, c in minus_cells)
        for k, (r, c) in enumerate(loop):
            allocation[r, c] += theta if k % 2 == 0 else -theta
        leaving = None                          # one minus-cell leaves basis
        for r, c in minus_cells:
            if abs(allocation[r, c]) < 1e-10:
                allocation[r, c] = 0.0; leaving = (r, c); break
        if leaving is not None: basic_cells.remove(leaving)
        basic_cells.append(entering); num_iterations += 1


    result_alloc = allocation[:m_orig, :n_orig]
    obj_value = float(np.sum(result_alloc * np.array(costs)[:m_orig, :n_orig]))
    return {"status": "optimal",
            "allocation": [[round(float(v), 6) for v in row] for row in
result_alloc.tolist()],
            "optimal_value": round(obj_value, 6),
            "num_iterations": num_iterations,
            "dummy_row": dummy_row, "dummy_col": dummy_col}
```

Лістинг А.5 — backend/core/validator.py (фрагмент)

Валідатори перевіряють дії студента й повертають уніфіковану структуру {valid, message, hint, ...} з підказками українською. Наведено дві репрезентативні функції; решта чотири (перерахунок таблиці, потенціали, вибір вхідної клітинки, побудова плану) побудовані аналогічно.

```
import math
import numpy as np
from typing import List, Dict, Any, Optional



def validate_simplex_pivot(tableau, col_names, user_pivot_col, user_pivot_row):
    """Validate a student's pivot selection against the correct simplex choice."""
    T = np.array(tableau, dtype=float)
    m = T.shape[0] - 1; n_total = T.shape[1] - 1
    obj_row = T[m, :n_total]


    if float(np.min(obj_row)) >= -1e-9:
        return {"valid": False,
                "message": "Таблиця вже оптимальна — зведення не потрібне.",
                "correct_col": None, "correct_row": None,
                "hint": "Усі коефіцієнти рядка цільової функції >= 0."}


    correct_col = int(np.argmin(obj_row))
    col_vals = T[:m, correct_col]; rhs = T[:m, -1]
    with np.errstate(divide="ignore", invalid="ignore"):
        ratios = np.where(col_vals > 1e-9, rhs / col_vals, np.inf)
    correct_row = int(np.argmin(ratios))
```

```
    if user_pivot_col == correct_col and user_pivot_row == correct_row:
        return {"valid": True,
                "message": "Правильно! Ведучий стовпець і рядок вибрані вірно.",
                "correct_col": correct_col, "correct_row": correct_row, "hint": None}


    if user_pivot_col != correct_col:
        name = col_names[correct_col]
        return {"valid": False,
                "message": "Неправильний ведучий стовпець. Оберіть змінну з найбільшим "
                           "відемним коефіцієнтом рядка цільової функції.",
                "correct_col": correct_col, "correct_row": correct_row,
                "hint": f"Найбільший відемний коефіцієнт — у стовпці «{name}»."}


    return {"valid": False,
            "message": "Неправильний ведучий рядок. Застосуйте правило мінімального "
                       "відношення theta = RHS / a_ij (лише для a_ij > 0).",
            "correct_col": correct_col, "correct_row": correct_row,
            "hint": f"Мінімальне theta — у рядку {correct_row + 1}."}



def validate_bnb_branch(lp_solution, user_branch_var):
    """Correct branching variable is the most fractional one (closest to 0.5)."""
    best_idx = None; best_dist = 0.0
    for i, val in enumerate(lp_solution):
        frac = val - math.floor(val)
        if frac > 1e-6:
            dist = min(frac, 1.0 - frac)
            if dist > best_dist: best_dist = dist; best_idx = i


    if best_idx is None:
        return {"valid": False,
                "message": "Всі змінні цілочисельні — розгалуження не потрібне.",
                "correct_var": None, "hint": None}


    if user_branch_var == best_idx:
        frac = lp_solution[best_idx] - math.floor(lp_solution[best_idx])
        return {"valid": True,
                "message": f"Правильно! x{best_idx + 1} найдробовіша (частка ~
{frac:.4f}).",
                "correct_var": best_idx, "hint": None}


    return {"valid": False,
            "message": "Неправильна змінна. Оберіть змінну з найбільшою дробовою "
                       "частиною (стратегія «most fractional»).",
            "correct_var": best_idx,
            "hint": f"Найдробовіша змінна — x{best_idx + 1}."}
```

Лістинг А.6 — backend/core/generator.py (фрагмент)

Генератор створює випадкові задачі з гарантованою допустимістю та нетривіальністю розв'язку. Наведено функцію для задач лінійного програмування; функції generate_ilp і generate_transport побудовані аналогічно (для ЦП додатково перевіряється, що LP-релаксація має дробовий розв'язок).

```
import random
from typing import Any, Dict, Optional
from scipy.optimize import linprog
```

```
def generate_lp(n_vars=2, n_constraints=3, maximize=True, seed=None, max_attempts=30):
    rng = random.Random(seed)
    for _ in range(max_attempts):
        c = [rng.randint(1, 9) for _ in range(n_vars)]
        A = [[rng.randint(1, 6) for _ in range(n_vars)] for _ in range(n_constraints)]
        b = [rng.randint(n_vars * 4, n_vars * 12) for _ in range(n_constraints)]


        c_min = [-v for v in c] if maximize else c[:]
        result = linprog(c_min, A_ub=A, b_ub=b, bounds=[(0, None)] * n_vars,
method="highs")
        if result.status != 0:
            continue
        if all(abs(v) < 1e-6 for v in result.x):   # skip trivial all-zero
            continue


        opt = -result.fun if maximize else result.fun
        return {"c": c, "A": A, "b": b, "maximize": maximize,
                "expected": {"optimal_value": round(opt, 4),
                             "solution": [round(v, 4) for v in result.x]}}
    return None
```

Лістинг А.7 — backend/api/routes.py (фрагмент)

Усі 18 ендпоінтів зібрано у Flask Blueprint api_bp з префіксом /api. Кожен суворо валідує вхідний JSON і повертає структуровану помилку. Наведено три типові маршрути — розв'язувач, валідатор та генератор; решта побудовані за тим самим шаблоном.

```
from flask import Blueprint, request, jsonify
from core.simplex_solver import solve_simplex
from core.validator import validate_simplex_pivot
from core.generator import generate_lp, generate_ilp, generate_transport


api_bp = Blueprint("api", __name__)



@api_bp.route("/simplex", methods=["POST"])
def simplex_endpoint():
    """Solve LP via simplex method, return step-by-step tableaux."""
    data = request.get_json(silent=True)
    if data is None:
        return jsonify({"error": "Очікується JSON"}), 400
    for field in ("c", "A", "b"):
        if field not in data:
            return jsonify({"error": f"Відсутнє поле: {field}"}), 400
    try:
        result = solve_simplex(c=data["c"], A=data["A"], b=data["b"],
                               maximize=bool(data.get("maximize", False)))
        return jsonify(result)
    except Exception as exc:
        return jsonify({"error": str(exc)}), 500



@api_bp.route("/simplex/check", methods=["POST"])
def simplex_check():
    """Validate student's pivot column/row choice."""
    data = request.get_json(silent=True)
    if data is None:
        return jsonify({"error": "Очікується JSON"}), 400
    try:
        result = validate_simplex_pivot(
            tableau=data["tableau"], col_names=data["col_names"],
            user_pivot_col=int(data["user_pivot_col"]),
            user_pivot_row=int(data["user_pivot_row"]))
        return jsonify(result)
```

```
    except Exception as exc:
        return jsonify({"error": str(exc)}), 500



@api_bp.route("/generate", methods=["GET"])
def generate_task():
    """Generate a random task of the specified type."""
    task_type = request.args.get("type", "simplex")
    n_vars = int(request.args.get("vars", 2))
    n_constraints = int(request.args.get("constraints", 3))
    if task_type == "simplex":
        problem = generate_lp(n_vars=n_vars, n_constraints=n_constraints)
    elif task_type == "branch_and_bound":
        problem = generate_ilp(n_vars=n_vars, n_constraints=n_constraints)
    elif task_type == "transport":
        problem = generate_transport(n_sources=n_vars, n_dests=n_constraints)
    else:
        return jsonify({"error": f"Невідомий тип задачі: {task_type}"}), 400
    if problem is None:
        return jsonify({"error": "Не вдалося згенерувати задачу"}), 500
    return jsonify({"type": task_type, "problem": problem})
```