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
«ПРОГРАМНА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ДЛЯ МОДЕЛЮВАННЯ ТА ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ЗАДАЧ»

зі спеціальності 122 «Комп’ютерні науки»

освітня програма «Комп’ютерні науки»

ступеня бакалавр

Виконавець роботи Лакатош Дар'я Миколаївна

_____ «___»____2026 р.
(підпис)

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

_____ «___»___2026 р.
(підпис)

Рецензент _____

ПОЛТАВА 2026

РЕФЕРАТ

Записка: 58 с., 15 рис., 2 таблиці, 2 додатки, 21 джерело.

ТРАНСПОРТНА ЗАДАЧА, ОПОРНИЙ ПЛАН, МЕТОД ПОТЕНЦІАЛІВ, ОПТИМІЗАЦІЯ ПЕРЕВЕЗЕНЬ, ЛІНІЙНЕ ПРОГРАМУВАННЯ, МЕТОД АПРОКСИМАЦІЇ ФОГЕЛЯ, PYTHON, TKINTER.

Об’єкт розробки – процес моделювання та оптимального розв’язування транспортних задач у програмному середовищі з покроковою візуалізацією обчислень.

Мета роботи – алгоритмізація та програмна реалізація елементів системи для моделювання і розв’язування транспортних задач із підтримкою класичних методів побудови опорного плану та оптимізації методом потенціалів.

Методи дослідження – методи дослідження операцій та лінійного програмування (метод північно-західного кута, метод мінімальної вартості, метод апроксимації Фогеля, метод потенціалів MODI), мова програмування Python та стандартна графічна бібліотека Tkinter.

Зроблено огляд існуючих програмних засобів для моделювання та розв’язування транспортних задач (Excel Solver, MATLAB, GAMS/AMPL, PuLP, Google OR-Tools), виявлено їхні переваги та недоліки. Розглянуто теоретичні основи транспортної задачі, методи побудови опорного плану та оптимізації методом потенціалів. Розроблено алгоритм роботи системи, побудовано його блок-схему.

Здійснена програмна реалізація системи мовою Python з використанням бібліотеки Tkinter для побудови графічного інтерфейсу. Реалізовано три методи побудови опорного плану, оптимізацію методом потенціалів (MODI), перевірку балансу задачі, обробку виродженості та покрокове ведення журналу обчислень. Програму зібрано у виконуваний файл (.exe) за допомогою PyInstaller.

В результаті тестування на прикладах різної складності (збалансована та незбалансована задачі, вироджений опорний план, порівняння методів побудови) виявлено, що система коректно знаходить оптимальний план перевезень, забезпечує

прозоре відображення усіх етапів обчислень і є зручною у використанні як у навчальному процесі, так і для прикладних задач планування перевезень.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	**Пояснення умовних позначень, скорочень, символів**
GUI	Graphical User Interface – графічний інтерфейс користувача
Least Cost	метод мінімальної вартості – метод побудови опорного плану, на кожному кроці якого обирається клітина з найменшою вартістю перевезення
MODI	Modified Distribution method – метод модифікованого розподілу (метод потенціалів), що використовується для оптимізації опорного плану транспортної задачі
NW corner	North-West corner – метод північно-західного кута побудови опорного плану
VAM	Vogel’s Approximation Method – метод апроксимації Фогеля для побудови опорного плану
Опорний план	початковий допустимий розподіл перевезень, що задовольняє обмеження за запасами й потребами; є вихідною точкою для подальшої оптимізації
m	кількість постачальників у транспортній задачі

n	кількість споживачів у транспортній задачі
a_i	обсяг запасу i-го постачальника
b_j	обсяг потреби j-го споживача
c_{ij}	вартість перевезення одиниці продукту з i-го пункту відправлення в j-й пункт призначення
x_{ij}	обсяг перевезення з i-го пункту відправлення в j-й пункт призначення (елемент плану перевезень)
u_i	потенціал i-го рядка в методі потенціалів (MODI)
v_j	потенціал j-го стовпця в методі потенціалів (MODI)
Δ_{ij}	оцінка вільної клітинки в методі потенціалів
Z	цільова функція загальної вартості перевезень
Σ	символ підсумовування (знак суми)

ВСТУП

Без математичних методів оптимізації сьогодні неможливо уявити роботу логістичних служб і управління ланцюгами постачання. Планування перевезень, розподіл ресурсів між складами, організація доставки, скорочення витрат на доставку – у кожній із цих задач використовуються підходи дослідження операцій. Класичним прикладом такого підходу є транспортна задача, у якій потрібно знайти, як розвезти однорідний продукт від постачальників до споживачів так, щоб сумарна вартість перевезень була мінімальною. Результати її розв'язування лягають в основу управлінських рішень про організацію вантажопотоків, і в умовах цифрової трансформації економіки роль таких рішень тільки зростає.

Ручні обчислення в задачах хоч трохи серйозного розміру швидко стають непрактичними, тому з'являється запит на програмні засоби, які будують і оптимізують плани перевезень самостійно. Сфера застосування таких систем – від планування доставки товарів між складами і магазинами до балансування виробничих потужностей; окремо варто згадати їхнє використання у викладанні дисциплін, пов'язаних із дослідженням операцій. При цьому за зовнішньою простотою стоїть досить чіткий математичний апарат, який потрібно акуратно перенести у код.

У межах цієї роботи розроблено програму для моделювання й розв'язування транспортних задач. В її основі лежать три класичні методи побудови опорного плану – північно-західного кута, мінімальної вартості та апроксимації Фогеля – і метод потенціалів (MODI), який доводить план до оптимального. Реалізація виконана мовою Python, інтерфейс побудовано на бібліотеці Tkinter: користувач задає обсяги постачання, попиту й матрицю вартостей, після чого може спостерігати весь хід обчислень та отриманий план перевезень. Для зручності поширення програму зібрано у виконуваний файл за допомогою PyInstaller.

Мета кваліфікаційної роботи – створити програмний засіб для моделювання та розв'язування транспортних задач, який поєднує реалізацію основних алгоритмів оптимізації з простим у користуванні інтерфейсом.

Для досягнення цієї мети у роботі вирішено такі завдання:

- проаналізувати наявні підходи до моделювання й розв'язування транспортних задач;
- розглянути методи побудови опорного плану та алгоритми його оптимізації;
- спроєктувати архітектуру системи та реалізувати її основні модулі на Python;
- побудувати графічний інтерфейс користувача засобами Tkinter;
- запровадити покрокове відображення процесу розв'язування;
- перевірити роботу системи на практичних прикладах;
- зібрати програму у виконуваний файл для зручного розповсюдження.

Об'єкт дослідження – процес моделювання та оптимального розв'язування транспортних задач.

Предмет дослідження – програмна реалізація методів побудови опорного плану та оптимізації перевезень, а також засоби візуалізації обчислень і способи формалізованого представлення вхідних даних.

У роботі застосовано методи дослідження операцій і лінійного програмування, прийоми алгоритмізації, структурного програмування й тестування програмного забезпечення.

Практичне значення роботи полягає в тому, що отримано наочний і простий у використанні інструмент, з яким можуть працювати студенти, дослідники та фахівці з логістики – як для вивчення самих методів, так і для побудови й оптимізації планів перевезень у власних задачах. Систему можна застосовувати у навчальному процесі, у науково-дослідній роботі, а також брати за основу для складніших систем планування та оптимізації.

Наукова новизна полягає в інтерактивному інтерфейсі, у якому користувач бачить кожен крок побудови й оптимізації плану – це робить процес обчислень прозорим і зручним для аналізу. Система орієнтована на освітнє використання і легко розширюється під нові задачі.

1. ПОСТАНОВКА ЗАДАЧІ

1.1. Актуальність задачі та теоретичне обґрунтування

Моделювання та розв'язування транспортних задач є однією з найвживаніших задач дослідження операцій: на ньому будується робота логістичних служб, керування ланцюгами постачання, виробниче планування, системи підтримки прийняття рішень. Що далі – то більше обсяги вантажоперевезень та складність логістичних мереж, і потреба в інструментах, здатних швидко й точно опрацювати такі задачі без виснажливих ручних обчислень, зростає. Ці інструменти мають уміти будувати допустимі плани перевезень, знаходити серед них найдешевший та оцінювати ефективність розподілу ресурсів.

Кількість прикладних задач, що зводяться до транспортної моделі, доволі велика. Серед типових:

- доставка однорідного товару зі складів у мережу магазинів;
- розподіл продукції між цехами виробництва та пунктами споживання;
- перевезення сировини від постачальників до переробних підприємств;
- скорочення сумарних витрат на логістику за заданих обмежень на постачання й попит.

За такими системами стоїть простий запит: позбавити користувача рутини ручних розрахунків і одночасно дати результат точніше та швидше. Особливо це важливо для сфер, де навіть невелике зниження питомих витрат на перевезення відчутно позначається на бюджеті підприємства – роздрібної торгівлі, дистрибуції, виробничої логістики. За масштабів великого підприємства такі заощадження здатні переростати в помітну економічну перевагу.

Звідси випливає, що автоматизація моделювання та розв'язування транспортних задач – невід'ємна частина сучасних інформаційних систем планування. Запропонована в роботі система розроблялась із прицілом на гнучкість, прозорість обчислень, інтерактивність та доступність для звичайного користувача.

Сама модель спирається на апарат лінійного програмування, окремим випадком якого і є класична транспортна задача. Вона описує співвідношення між

обсягами постачання, попиту та вартостями перевезення одиниці продукції на кожному з можливих маршрутів «постачальник – споживач».

Якщо формалізувати, модель транспортної задачі складається з таких компонентів:

- вектор обсягів постачання (supply) для кожного постачальника;
- вектор обсягів попиту (demand) для кожного споживача;
- матриця вартостей (cost) перевезення одиниці продукції;
- матриця плану перевезень, у якій і записуються шукані обсяги поставок.

У програмі ці елементи представлені звичайними структурами даних Python – списками й вкладеними списками. Завдяки цьому код виходить модульним, простим у підтримці та готовим до повторного використання.

Окремо реалізовано:

- перевірку балансу задачі (закрита або відкрита) з додаванням фіктивного постачальника чи споживача;
- три методи побудови початкового опорного плану – північно-західного кута, мінімальної вартості та апроксимації Фогеля;
- оптимізацію плану методом потенціалів (MODI);
- покрокове логування ходу обчислень;
- контроль виродженості плану, без якого MODI коректно не працює.

Усе перелічене формує методологічну основу, на якій будується практична частина роботи.

1.2. Практична постановка задачі та характеристики системи

Мета роботи – створити програмну систему для моделювання та розв'язування транспортних задач: зручну в користуванні, з прозорою архітектурою і повним циклом розв'язування – від уведення вихідних даних до отримання оптимального плану з покроковим поясненням обчислень.

У межах цієї мети поставлено такі задачі:

- Спроєктувати внутрішню модель задачі, у якій будуть представлені обсяги постачання, попиту й матриця вартостей.

- Реалізувати три методи побудови початкового опорного плану – північно-західного кута, мінімальної вартості та апроксимації Фогеля.
- Запрограмувати оптимізацію опорного плану методом потенціалів (MODI) з обчисленням оцінок вільних клітинок.
- Побудувати графічний інтерфейс, у якому користувач уводить дані, обирає метод побудови плану, запускає розв'язування й бачить усі його етапи.
- Додати журнал кроків, щоб користувач міг візуально контролювати кожен етап обчислень.
- Витримати модульну архітектуру коду, яка дає простір для розширення – наприклад, додавання нових методів побудови плану або критеріїв оптимізації.
- Підготувати систему до збирання у виконуваний файл (.exe), щоб її можна було поширювати без додаткових залежностей.

Реалізація виконується мовою Python; для графічного інтерфейсу обрано Tkinter – стандартну бібліотеку, з якою можна швидко зібрати простий і зрозумілий GUI. Основні модулі програми:

- структури для зберігання вхідних даних (вектори постачання та попиту, матриця вартостей);
- функції перевірки балансу й зведення задачі до закритого вигляду;
- модулі побудови опорного плану та оптимізації методом потенціалів;
- система ведення журналу кроків;
- інтерфейсна частина з усією взаємодією користувача.

Очікуваний результат – інтерактивна програма, у якій користувач задає умови транспортної задачі, аналізує процес її розв'язування, експериментує з різними вхідними даними й отримує оптимальний план перевезень. Такий інструмент знайде застосування і в навчанні, і в прикладному плануванні перевезень.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Аналіз існуючих систем для моделювання та розв'язування транспортних задач

Для моделювання й розв'язування транспортних задач існує чимало програмних засобів та систем оптимізації – від наукових розробок до комерційних продуктів і навчальних інструментів. Вони відрізняються тим, як формалізують задачу, наскільки добре масштабуються та який інтерфейс пропонують користувачеві. Автоматизація, у свою чергу, не лише прискорює пошук оптимального розподілу перевезень, а й суттєво зменшує ймовірність помилок порівняно з ручними обчисленнями. Нижче коротко розглянуто найвживаніші з таких засобів.

Microsoft Excel із надбудовою «Пошук розв'язку» (Solver) – мабуть, найдоступніший інструмент для розв'язування невеликих транспортних задач: цільова функція й обмеження задаються прямо в клітинках таблиці. Найчастіше його використовують у навчанні та для нескладних прикладних задач. До переваг належать простота, поширеність і наочність; до недоліків – погана робота з великими задачами та відсутність прозорого покрокового виведення алгоритму.

MATLAB разом з Optimization Toolbox – потужне середовище для чисельних розрахунків, у якому підтримується лінійне програмування і транспортні моделі. Перевагами є висока швидкодія, багатий математичний апарат і набір готових функцій оптимізації; обмеженнями – комерційна ліцензія та надмірна для суто навчальних задач складність.

GAMS та AMPL – спеціалізовані мови алгебраїчного моделювання, які призначені насамперед для формулювання задач оптимізації великої розмірності. Вони пропонують гнучке моделювання й інтеграцію з різноманітними розв'язувачами, але потребують серйозного часу на освоєння і коштовних ліцензій.

PuLP – бібліотека для Python, у якій транспортну задачу можна описати в коді у вигляді цільової функції та лінійних обмежень, а сам пошук розв'язку делегувати зовнішньому LP-розв'язувачу. Бібліотека безкоштовна і добре інтегрується з рештою Python-екосистеми, проте не має вбудованих засобів для візуалізації проміжних кроків.

Google OR-Tools – бібліотека з відкритим кодом для задач оптимізації та комбінаторики з ефективними розв'язувачами і підтримкою кількох мов програмування. На відміну від перелічених засобів, запропонована в роботі система робить ставку не на потужність, а на поєднання простого введення даних з повністю прозорим покроковим відображенням побудови й оптимізації плану. Її головна перевага – навчальна спрямованість і можливість бачити кожен крок розв'язування, що корисно як у викладанні, так і в самостійному опрацюванні методів дослідження операцій.

Коротка порівняльна характеристика цих систем зведена в таблиці 2.1.

Таблиця 2.1 – Огляд існуючих засобів для розв'язування транспортних задач

Назва	**Категорія**	**Особливості**	**Переваги**	**Недоліки**
Excel Solver	Табличний оптимізатор	Розв'язування через надбудову	Простота, поширеність	Обмежена розмірність
MATLAB	Математичне середовище	Optimization Toolbox	Продуктивність, апарат	Платність, складність
GAMS/AMPL	Мови моделювання	Опис задач оптимізації	Гнучкість, інтеграція	Високий поріг входу
PuLP	Бібліотека Python	Опис LP-задач у коді	Безкоштовність, інтеграція	Немає візуалізації

OR-Tools	Бібліотека оптимізації	Відкритий код, швидкодія	Ефективність, підтримка	Складність для новачків

2.2. Аналіз методів розв'язування транспортної задачі

Розв'язування транспортної задачі традиційно поділяється на два етапи: побудову початкового опорного плану і його подальшу оптимізацію. Спершу шукається будь-який допустимий розподіл перевезень, а потім він покроково поліпшується до оптимального за критерієм мінімальної загальної вартості.

Опорний план будують одним із трьох класичних методів:

- метод північно-західного кута (NW corner) – найпростіший: клітини заповнюються послідовно з верхнього лівого кута, без огляду на вартості перевезень;
- метод мінімальної вартості (Least Cost) – на кожному кроці обирається клітина з найменшою вартістю, що, як правило, дає кращий за NW початковий план;
- метод апроксимації Фогеля (VAM) – використовує штрафи, тобто різниці двох найменших вартостей у рядку чи стовпці, і зазвичай дає опорний план, найближчий до оптимального.

Наприклад, у рядку з вартостями (4, 6) штраф дорівнює $6 - 4 = 2$, і перевагу отримує той рядок чи стовпець, де штраф найбільший.

Для оптимізації опорного плану використовують метод потенціалів (MODI) або еквівалентний йому метод стрибання по клітинах (stepping-stone). Кожній вільній клітинці присвоюється оцінка: якщо хоча б одна з них від'ємна – план ще можна поліпшити, перерозподіливши постачання вздовж замкненого циклу.

Серед переваг наведених методів варто відзначити:

- формальну коректність і гарантовану збіжність до оптимального плану;

- нескладну реалізацію базових методів побудови;
- наявність зрозумілих кроків, які можна логувати й контролювати.

Недоліки теж очевидні:

- північно-західний кут ігнорує вартості, тож стартовий план буває далеким від оптимального;
- план може виявитися виродженим – це потребує додаткової обробки;
- обсяг обчислень швидко зростає зі збільшенням розмірності задачі.

У системі оптимізація виконується послідовним обчисленням потенціалів та оцінок вільних клітин з фіксацією кожного кроку в журналі. Отже, у розробці враховано обидва етапи – побудову опорного плану і його доведення до оптимуму – з огляду на ключові властивості та ефективність відповідних методів.

2.3. Технології для реалізації системи

Програмне забезпечення розроблено мовою Python. Вибір на її користь зумовлений популярністю мови, високою швидкістю розробки, підтримкою об'єктно-орієнтованого підходу та великим набором вбудованих засобів для роботи зі структурами даних, матрицями і графічним інтерфейсом.

Інтерфейсна частина побудована на Tkinter – стандартній бібліотеці для створення графічних інтерфейсів у Python. З Tkinter можна швидко зібрати віконний додаток із кнопками, полями введення, таблицями та мітками. Він є частиною стандартної поставки Python, що робить його зручним вибором для навчального ПЗ. Можливостей цієї бібліотеки виявилося цілком достатньо для всіх запланованих функцій: введення обсягів постачання й попиту, формування матриці вартостей, виведення проміжних кроків і відображення оптимального плану.

Окремо варто згадати PyInstaller – утиліту, яка перетворює Python-програму на виконуваний файл. Це дало змогу зібрати незалежний додаток, для запуску якого не потрібно мати встановлений інтерпретатор Python. Особливо це зручно для поширення програми серед студентів, викладачів чи інших користувачів без спеціальних технічних навичок.

Крім того, у проєкті задіяні:

- copy – стандартний модуль Python для глибокого копіювання матриць плану під час збереження стану на кожному кроці розв'язування;
- collections – структури даних, які стали в пригоді для організації проміжних обчислень і пошуку циклів перерозподілу;
- ScrolledText – компонент Tkinter, у якому зручно показувати великі обсяги тексту, як-от журнал кроків побудови й оптимізації плану.

Обраний стек технологій дозволив реалізувати повноцінну систему для моделювання та розв'язування транспортних задач – зручну у використанні, легку в розповсюдженні й цілком придатну до поставлених задач. Її можна застосовувати як у навчальному процесі, так і як самостійний інструмент для ознайомлення з методами дослідження операцій.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Теоретичні основи транспортної задачі

Транспортна задача є окремим випадком задачі лінійного програмування і полягає в оптимальному розподілі однорідного продукту від постачальників до споживачів за умови мінімальної загальної вартості перевезень. У формальному вигляді вона описує співвідношення між обсягами постачання, попиту та вартостями перевезення одиниці продукції на кожному маршруті «постачальник – споживач». Це одна з класичних задач дослідження операцій, і завдяки специфіці своєї структури вона має ефективні спеціалізовані методи розв'язування, які працюють швидше за загальний симплекс-метод.

Базові поняття транспортної задачі такі:

- Постачальник – пункт, що має певний обсяг продукту для відправлення (запас a_i); наприклад, склад або виробниче підприємство.
- Споживач – пункт, якому потрібна певна кількість продукту (потреба b□); типові приклади – магазин або точка збуту.
- Вартість перевезення (c_i□) – витрати на доставку одиниці продукту з i-го пункту відправлення в j-й пункт призначення.
- План перевезень – матриця x_i□, у якій записано, скільки одиниць продукту йде кожним маршрутом.

У математичній постановці задачі потрібно мінімізувати цільову функцію загальної вартості:

$$Z = \Sigma\Sigma\, c_i\square \cdot x_i\square \rightarrow min,$$

за таких обмежень:

- $\Sigma\square\, x_i\square = \boldsymbol{a_i}$ – весь запас кожного постачальника має бути вивезений;
- $\Sigma_i\, x_i\square = b\square$ – потреба кожного споживача має бути задоволена;

- $x_i \geq 0$ - обсяги перевезень не можуть бути від'ємними.

Окремо стоїть умова балансу: коли сумарні запаси дорівнюють сумарним потребам ($\Sigma a_i = \Sigma b$), задача називається закритою або збалансованою. Якщо ж рівність не виконується, маємо відкриту задачу; для її розв'язування додають фіктивного постачальника або споживача з нульовими вартостями, чим і зводять задачу до закритого вигляду. Кількість допустимих планів навіть для невеликих задач велика, тому оптимум знаходять спеціальними методами побудови та покращення плану, а не повним перебором.

Приклад транспортної таблиці наведено в таблиці 3.1.

Таблиця 3.1 – Приклад транспортної таблиці (вартості перевезень, обсяги постачання та попиту)

	Споживач B_1	**Споживач B_2**	**Споживач B_3**	**Запаси (a_i)**
Постачальник A_1	4	6	8	30
Постачальник A_2	5	3	7	40
Постачальник A_3	6	2	4	20
Потреби (b)	20	50	20	90

У наведеному прикладі сумарні запаси (30 + 40 + 20 = 90) збігаються з сумарними потребами (20 + 50 + 20 = 90), тож задача збалансована. У кожній клітині таблиці записано вартість перевезення одиниці продукту відповідним маршрутом, а шуканий план перевезень визначає обсяги, що ці клітини заповнюють.

Допустимий план будується у два кроки:

- спершу формується опорний план – будь-який допустимий розподіл, що задовольняє обмеження за запасами й потребами;

- потім цей план послідовно покращується, доки не буде досягнуто оптимуму.

Опорний план називається невиродженим, якщо кількість заповнених (базисних) клітинок дорівнює $m + n - 1$, де m – кількість постачальників, а n – кількість споживачів. Виконання цієї умови необхідне для коректної роботи методу оптимізації; якщо її порушено, до плану додається нульове базисне перевезення.

Транспортну задачу варто розглядати не лише як абстрактну математичну модель, а й як основу прикладних обчислювальних систем планування. У реальних логістичних задачах постачальниками й споживачами виступають склади, виробничі майданчики та точки збуту, а матриця вартостей описує транспортні витрати, відстані або час доставки. Завдяки цьому одна й та сама модель підходить для широкого кола завдань – від планування доставки товарів до балансування виробничих потужностей і оптимізації маршрутів між підрозділами підприємства. Зміна вартостей або обсягів постачання й попиту дає змогу моделювати різні сценарії та оцінювати їхній вплив на сумарні витрати – це робить транспортну модель зручним інструментом аналізу управлінських рішень.

3.2. Алгоритм побудови та оптимізації плану перевезень

Алгоритми побудови опорного плану та його оптимізації – фундамент розв'язування транспортної задачі. У сукупності вони дозволяють знайти допустимий розподіл перевезень і поступово довести його до оптимального за критерієм мінімальної загальної вартості.

Побудова опорного плану зводиться до формування першого допустимого розподілу, який задовольняє обмеження за запасами та потребами. У методі північно-західного кута, наприклад, клітини послідовно заповнюються з верхнього лівого кута: у кожну записується максимально можливий обсяг, після чого вичерпаний рядок або стовпець виключається з розгляду.

Алгоритм побудови опорного плану складається з таких кроків:

- вибір чергової клітини за обраним методом (NW corner, Least Cost або VAM);
- запис у клітину максимально можливого обсягу перевезення;
- коригування залишкових запасів та потреб;
- повторення попередніх дій, доки не буде розподілено весь обсяг продукту.

Оптимізація плану виконується методом потенціалів (MODI). Для базисних клітинок обчислюються потенціали рядків (u_i) та стовпців (v□), а для вільних – оцінки Δ_i□ = c_i□ − (u_i + v□). Якщо всі Δ_i□ невід'ємні, план оптимальний; у протилежному випадку його покращують перерозподілом обсягів уздовж замкненого циклу.

Псевдокод оптимізації має вигляд:

```
while план не оптимальний:

  обчислити потенціали u_i, v□ для базисних клітин
  обчислити оцінки Δ_i□ для вільних клітин
  якщо всі Δ_i□ ≥ 0:
    план оптимальний
  інакше:
    обрати клітину з найменшою оцінкою
    побудувати замкнений цикл перерозподілу
    перерозподілити обсяги вздовж циклу
    зафіксувати крок у журналі
```

Кожен крок системи фіксується в журналі розв'язування, тож користувач може стежити за ходом обчислень. У навчальному контексті це особливо цінно, адже там важливий не лише сам результат, а й розуміння того, як план будувався і поступово покращувався.

У парі ці алгоритми ефективно розв'язують транспортну задачу, формалізують процес оптимізації та забезпечують математично обґрунтовану основу обчислювального модуля системи.

3.3. Блок-схема алгоритму роботи системи

Блок-схема на рисунку 3.1 ілюструє загальну логіку взаємодії користувача з системою та внутрішні етапи обробки задачі. Кожному етапу відповідає окрема функціональність у коді Python, а їх послідовність формує цілісний процес розв'язування від першого введення даних до отримання оптимального плану.

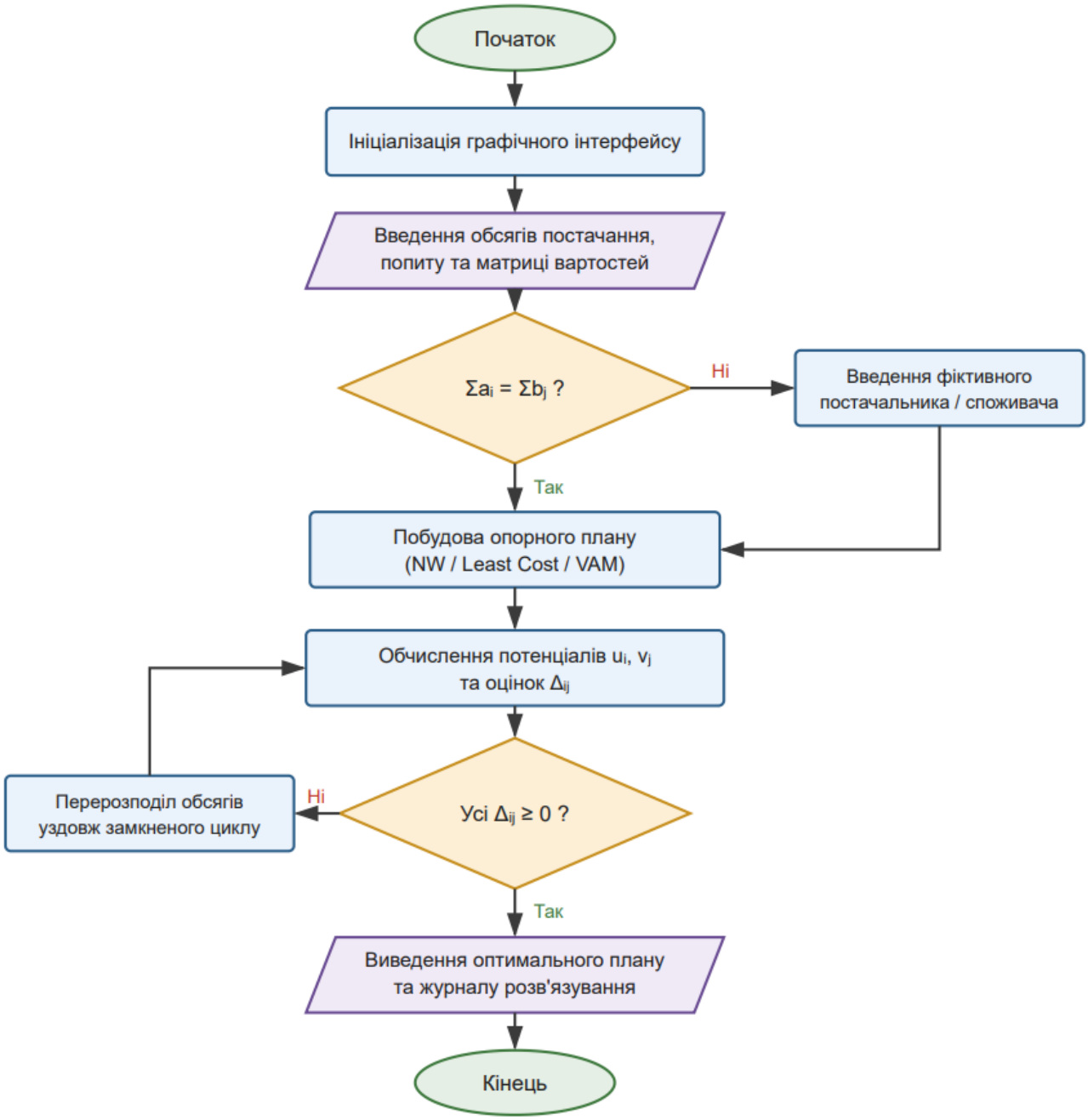

Рисунок 3.1 – Блок-схема роботи програми

Етапи схеми такі:

- Початок: запуск програми та ініціалізація графічного інтерфейсу.
- Введення обсягів постачання, попиту й матриці вартостей у відповідних полях інтерфейсу.
- Перевірка балансу: порівняння сумарних запасів і потреб, за потреби – додавання фіктивного постачальника чи споживача.
- Побудова опорного плану обраним методом (північно-західного кута, мінімальної вартості або апроксимації Фогеля).
- Запуск циклу оптимізації методом потенціалів (MODI):
- обчислення потенціалів і оцінок вільних клітинок;
- перевірка плану на оптимальність;
- перерозподіл обсягів уздовж замкненого циклу;
- фіксація кроку в журналі.
- Перевірка оптимальності (усі оцінки невід'ємні) – вихід із циклу.
- Виведення оптимального плану та журналу в інтерфейсі.
- Завершення роботи або повернення до нового введення.

Така схема дає чітке уявлення про структуру системи, логіку переходів між етапами і про те, як саме працює механізм розв'язування транспортної задачі.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Інструменти та середовище розробки

Розробка програмного забезпечення для моделювання та розв'язування транспортних задач – це не лише вибір алгоритмічного підходу, а й обдуманий підбір інструментів реалізації, від яких залежать зручність, портативність і масштабованість майбутньої системи. У цьому розділі обґрунтовано вибір мови програмування, бібліотеки для побудови інтерфейсу та середовища розробки.

Вибір мови програмування – Python

Python обрано основною мовою реалізації проєкту з кількох причин:

- популярність і простий синтаксис дозволяють зосередитися на логіці, а не на технічних деталях;
- підтримка об'єктно-орієнтованого програмування зручна для моделювання структур транспортної задачі – матриці вартостей, векторів постачання й попиту, матриці плану;
- мова має багато вбудованих засобів для роботи зі структурами даних, обчисленнями та інтерфейсом;
- активна спільнота забезпечує доступ до документації, прикладів і готових рішень.

Вибір бібліотеки для GUI – Tkinter

Tkinter – стандартна бібліотека для створення графічних інтерфейсів у Python. На користь її вибору промовляє кілька факторів:

- Стандартна поставка. Tkinter входить у стандартний дистрибутив Python – не потрібно нічого додатково встановлювати, програма готова до запуску одразу після інсталяції Python.

- Простота використання. Інтерфейс будується мінімалістичним і зрозумілим кодом, тож основну увагу можна приділити логіці системи, а не оформленню GUI.
- Кросплатформеність. Додатки на Tkinter без жодних змін у коді працюють на Windows, Linux і macOS.
- Документація та приклади. Багата база знань і активна спільнота дозволяють швидко знаходити розв'язання типових проблем, що виникають під час розробки.
- Повна інтеграція з Python. GUI легко з'єднується з обчислювальним модулем у межах однієї мови.

Вибір середовища розробки – Visual Studio Code

Середовищем розробки обрано Visual Studio Code (VS Code), яке має такі переваги:

- Відкритість і безкоштовність. VS Code розповсюджується під ліцензією MIT і доступний безкоштовно.
- Розширення. Доступна велика кількість плагінів – для Python, автоформатування, підсвічування синтаксису, запуску коду тощо.
- Кросплатформеність. VS Code однаково зручно працює на Windows, Linux та macOS.
- Інтеграція з Git. Вбудовані інструменти контролю версій спрощують ведення історії змін.
- Автодоповнення й аналіз коду. Це безпосередньо підвищує і швидкість, і точність розробки.

Зв'язка Python + Tkinter + Visual Studio Code дає оптимальний баланс між зручністю, гнучкістю, функціональністю й простотою реалізації. Такий вибір дозволяє зосередитися на основній логіці розв'язування, не витрачаючи зайвих зусиль на технічні налаштування. Окремо варто відзначити, що ці інструменти добре підходять для навчальних і демонстраційних цілей – а саме на таке

застосування системи в освітньому процесі чи самостійному вивченні методів дослідження операцій робиться головна ставка.

4.2. Розробка програмного забезпечення та графічного інтерфейсу

Програмне забезпечення реалізовано мовою Python із застосуванням структурованого підходу, що передбачає чіткий поділ на логічні модулі. Основною метою було забезпечити прозорість, модульність і зручність підтримки коду. Архітектура системи відокремлює обчислювальне ядро (перевірка балансу, побудова опорного плану, оптимізація методом потенціалів) від частини інтерфейсу (GUI).

Загальна структура реалізації включає такі основні компоненти:

- Опис структур даних: вектори обсягів постачання й попиту, матриця вартостей та матриця плану перевезень. Це дозволяє оперувати вихідними даними транспортної задачі у зручній для програмування формі.
- Перевірка балансу: функція, що порівнює сумарні запаси й потреби та за потреби вводить фіктивного постачальника або споживача, зводячи задачу до закритого вигляду.
- Побудова опорного плану: функції методів північно-західного кута, мінімальної вартості та апроксимації Фогеля, які формують початковий допустимий розподіл перевезень.
- Оптимізація: функції обчислення потенціалів та оцінок вільних клітин, пошуку замкненого циклу й перерозподілу обсягів реалізують метод потенціалів (MODI) із трасуванням усіх кроків – це дозволяє користувачеві спостерігати процес покращення плану.

Інтерфейс користувача побудовано за допомогою Tkinter, стандартної бібліотеки Python для створення графічних віконних застосунків. Вибір Tkinter обумовлено його простотою, відсутністю зовнішніх залежностей та

кросплатформенністю. GUI було спроєктовано з урахуванням зручності для користувача, особливо – у навчальному контексті.

Ключові елементи інтерфейсу:

- Інструкція користувача: на початку вікна програми наведено короткі правила введення даних – задання кількості постачальників і споживачів, заповнення матриці вартостей та векторів запасів і потреб.
- Поля введення даних: матриця вартостей подається у вигляді таблиці полів вводу, а обсяги постачання й попиту – окремими полями біля відповідних рядків і стовпців.
- Панель автоматизації: розташована над таблицею введення даних і містить випадальний список «Приклади» з готовими задачами (збалансована 2×2, 3×3, 3×4 та відкрита 2×3), кнопку «Завантажити приклад» для автоматичного заповнення таблиці, кнопку «Згенерувати задачу» для створення випадкової транспортної задачі, а також кнопки «Порівняти методи» та «Зберегти звіт».
- Вибір методу: перемикач, що дозволяє обрати метод побудови опорного плану (північно-західного кута, мінімальної вартості або апроксимації Фогеля).
- Кнопка «Розв'язати»: запускає процес розв'язування з відображенням усіх кроків у журналі.
- Журнал розв'язування: реалізований за допомогою компонента ScrolledText із моноширинним шрифтом Consolas, що забезпечує охайне вирівнювання матриць плану на кожному кроці. Містить текстовий протокол усіх етапів розв'язування з поясненнями обчислень потенціалів, оцінок та перерозподілів. На самому початку журналу зазначається тип задачі (закрита чи відкрита), який програма визначає автоматично за співвідношенням сумарних запасів і потреб.
- Результат: оптимальний план перевезень відображається у вигляді окремої таблиці з обрамленими клітинками та підписами рядків (постачальники A1, A2, …) і стовпців (споживачі B1, B2, …). Під таблицею виводиться рядок «Загальна вартість: …» зі значенням сумарної вартості перевезень. У журналі

додатково виводиться вартість опорного плану до оптимізації, що дозволяє оцінити, наскільки метод потенціалів покращив початкове рішення.

Такий підхід до організації інтерфейсу забезпечує інтуїтивну зрозумілість навіть для користувачів без попереднього досвіду роботи з транспортними задачами.

Нижче подано візуальні приклади ключових елементів інтерфейсу та приклад використання програми в різних сценаріях. Це дає змогу краще зрозуміти принципи роботи системи та взаємодії користувача з нею.

На рис. 4.1 зображено головне вікно програми з уведеними даними транспортної задачі. Чітко видно таблицю матриці вартостей, поля обсягів постачання й попиту, перемикач методу, кнопку «Розв'язати» та журнал.

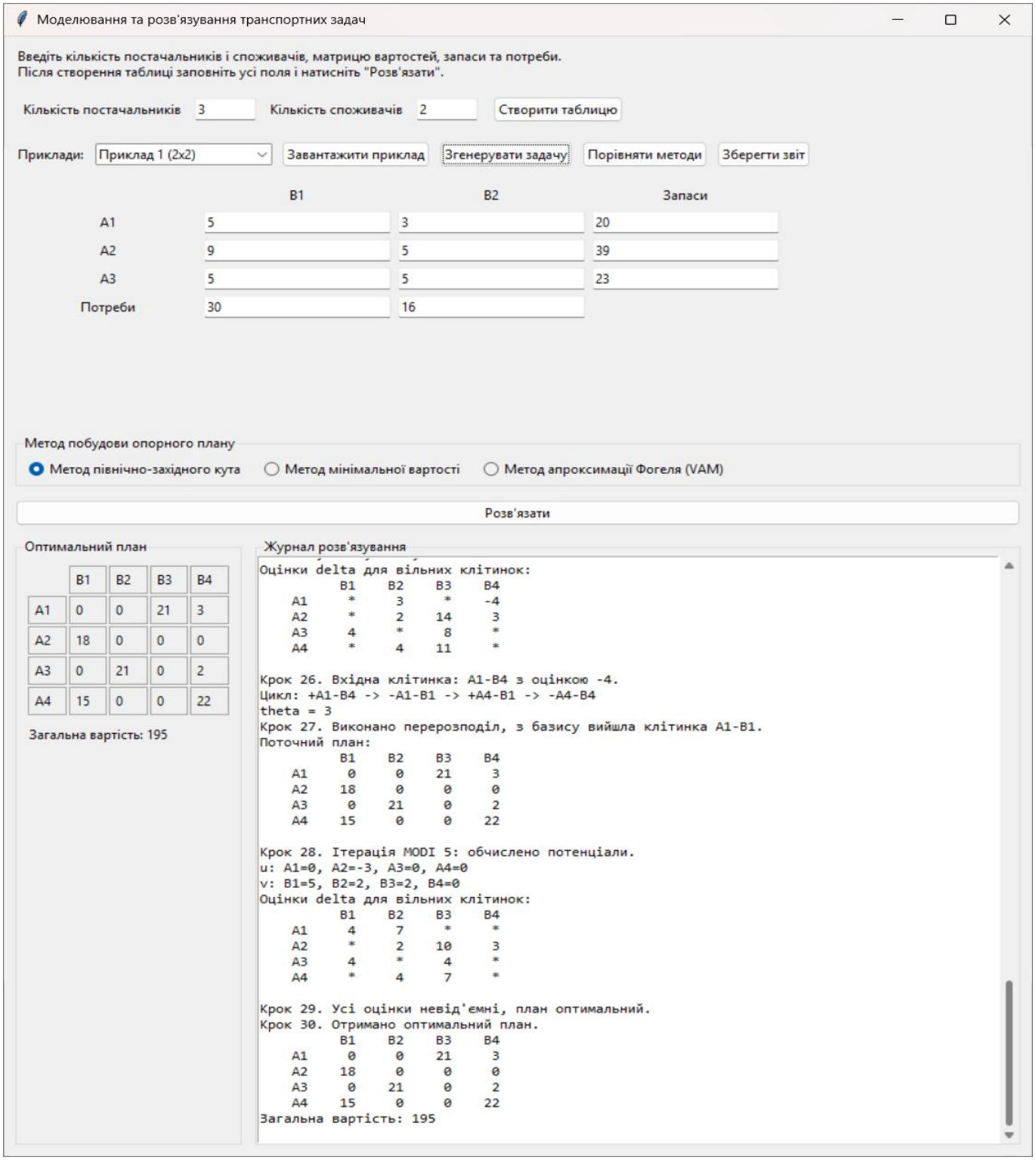

Рисунок 4.1 – Інтерфейс програми з даними

Вибір методу побудови плану:

На рис. 4.2 виділено, як користувач обирає метод побудови опорного плану за допомогою перемикача. Це дозволяє швидко порівнювати різні підходи до розв'язування.

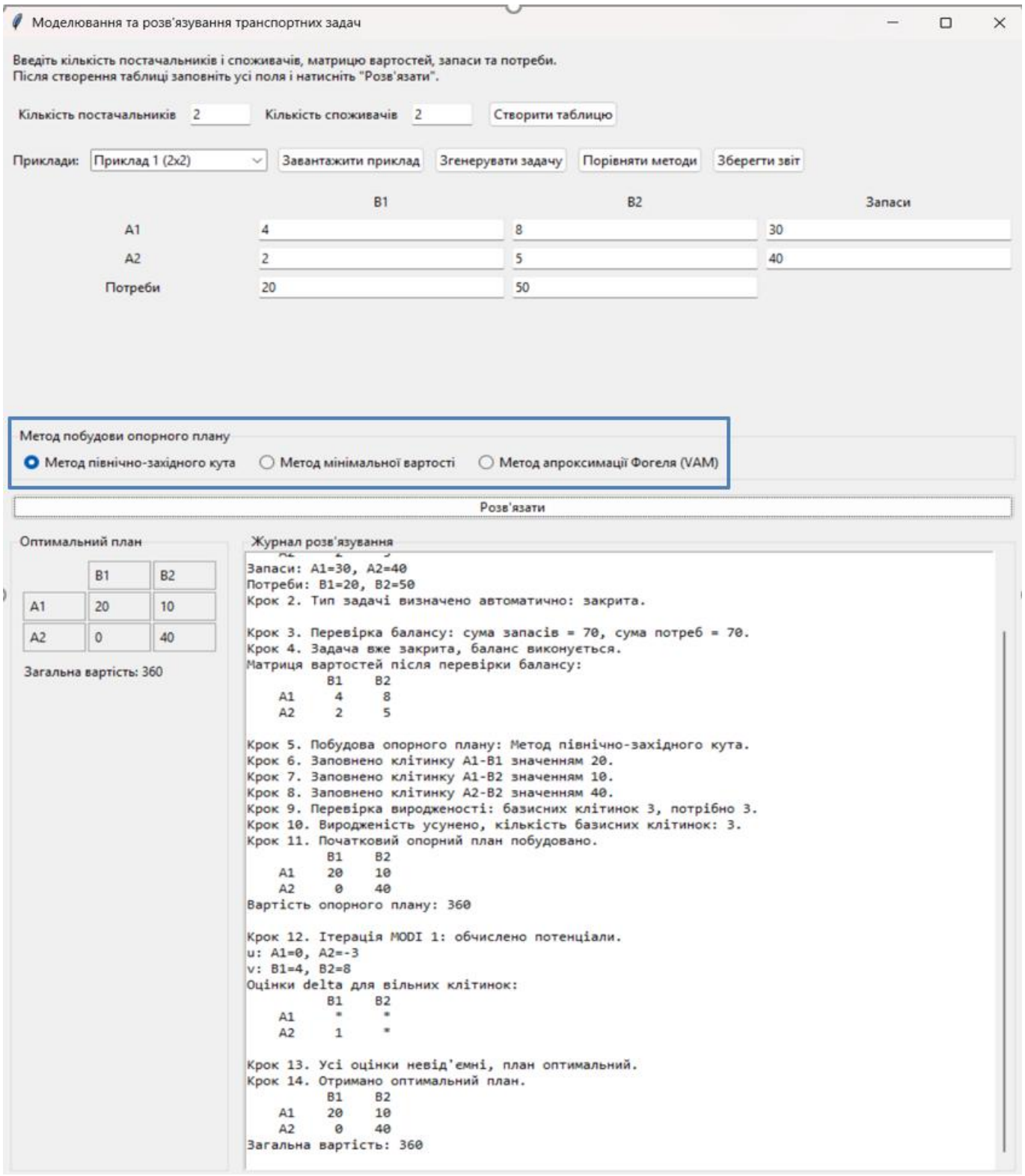

Рисунок 4.2 – Інтерфейс програми з виділеним функціоналом вибору методу

Приклад результату розв'язування:

Рис. 4.3 демонструє результат натискання кнопки «Розв'язати». У журналі показано:

- побудований опорний план;

- обчислені потенціали та оцінки вільних клітин;
- фінальний результат – оптимальний план та загальну вартість перевезень.

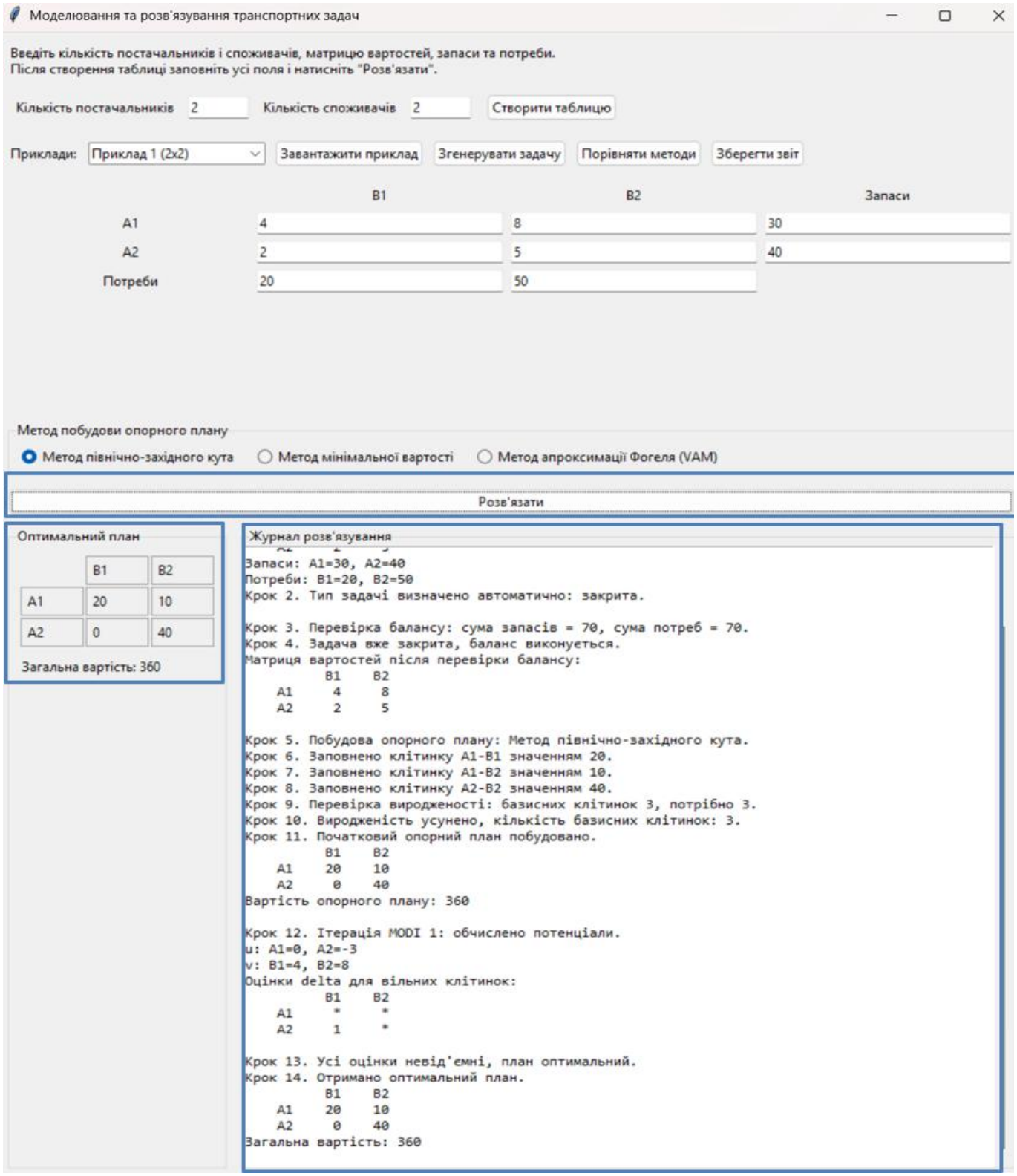

Рисунок 4.3 – Інтерфейс програми з виділеним функціоналом запуску, журналом розв'язування і результатом

На рис. 4.4 показано панель автоматизації, що розташована над таблицею введення даних, з випадальним списком готових прикладів та чотирма кнопками — для завантаження прикладу, генерації випадкової задачі, порівняння методів і збереження звіту.

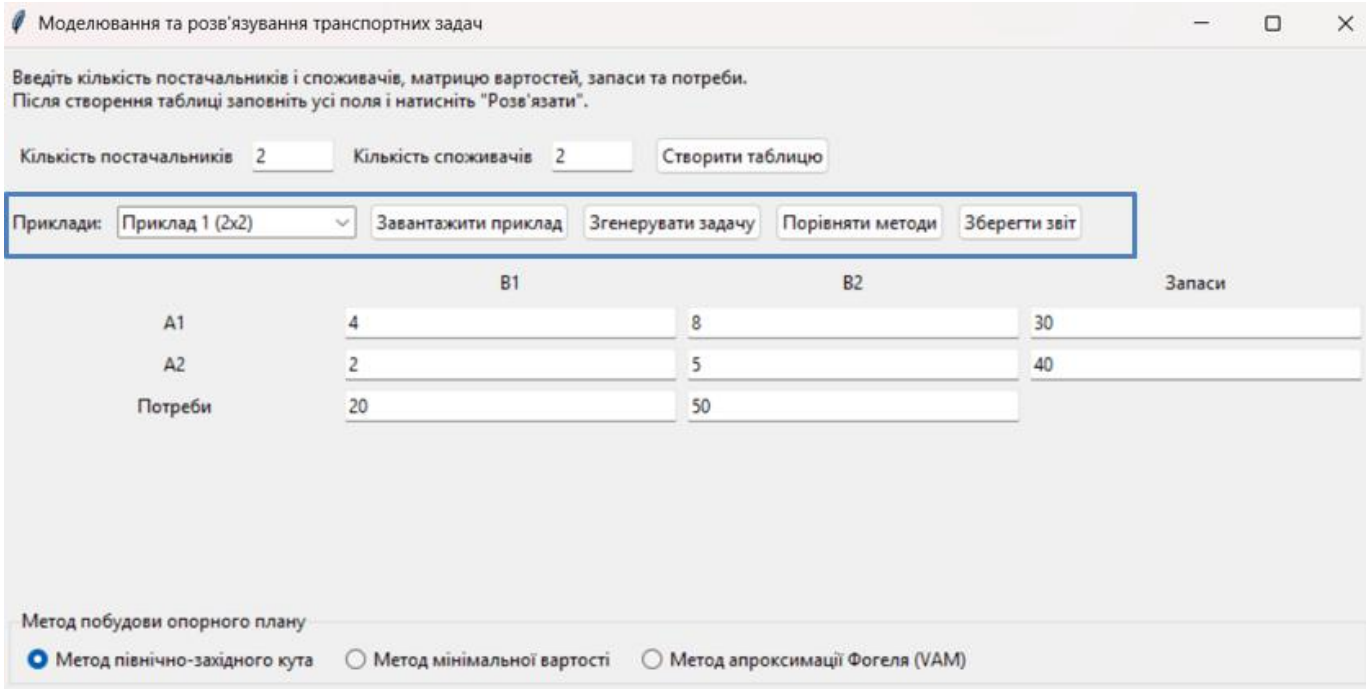

Рисунок 4.4 – *Панель автоматизації з готовими прикладами, генератором задач та інструментами порівняння і збереження результату*

Рис. 4.5 ілюструє результат натискання кнопки «Порівняти методи»: у журналі формується підсумкова таблиця, у якій для кожного з трьох методів зазначено вартість опорного плану та оптимальну вартість. Видно, що метод апроксимації Фогеля часто дає опорний план, який уже збігається з оптимальним, тоді як північно-західний кут потребує додаткових ітерацій оптимізації.

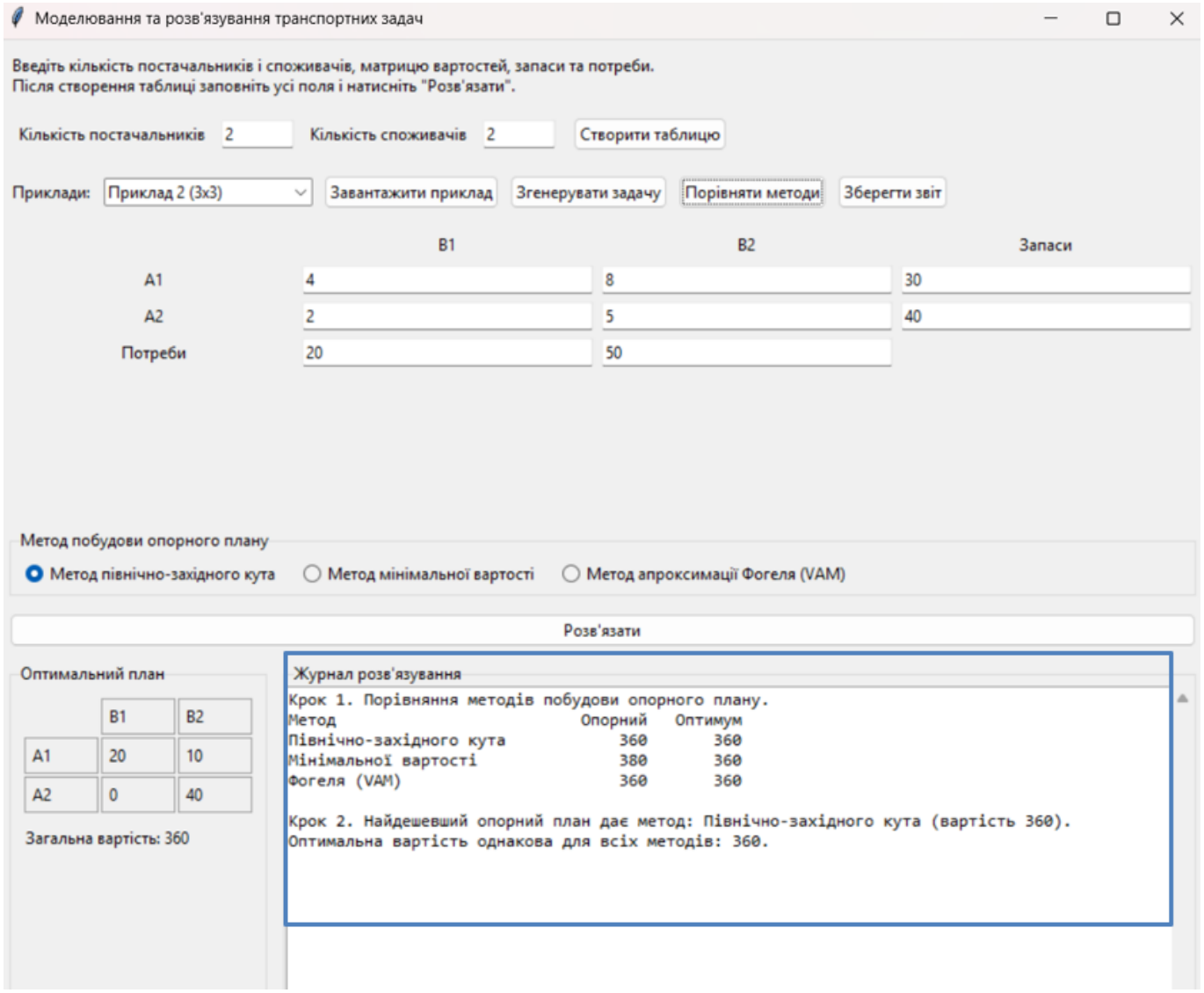

Рисунок 4.5 – *Результат роботи функції «Порівняти методи»: підсумкова таблиця у журналі розв'язування з вартостями опорних планів та оптимуму для кожного з трьох методів*

Завдяки такій реалізації інтерфейс є інтуїтивно зрозумілим навіть для користувачів без досвіду роботи з транспортними задачами або програмуванням.

Для зручності користувача в системі реалізовано додаткові функції автоматизації. Кнопка «Порівняти методи» поспіль запускає всі три методи побудови опорного плану на одних і тих самих вхідних даних та виводить у журналі підсумкову таблицю порівняння — метод, вартість опорного плану та оптимальна вартість, — окремо виділяючи метод із найдешевшим стартовим планом. Кнопка

«Зберегти звіт» дозволяє експортувати у текстовий файл (.txt) оптимальний план, загальну вартість і повний журнал розв'язування, що зручно для подальшого аналізу або обміну результатами. Завантаження готових прикладів та генератор випадкових задач, у свою чергу, спрощують перевірку роботи системи на різних типах вхідних даних без ручного введення.

4.3. Створення виконуваного файлу

Щоб зробити програму придатною до використання на будь-якому комп'ютері без необхідності встановлення Python та залежностей, було прийнято рішення зібрати виконуваний файл (.exe) за допомогою інструменту PyInstaller.

PyInstaller – це утиліта, яка дозволяє пакувати Python-програми у самодостатні виконувані файли. Такий файл містить інтерпретатор, усі модулі, ресурси, а також сам код, що дозволяє запускати програму навіть на машинах, де не встановлено Python.

Покроковий процес створення виконуваного файлу:

- Встановлення PyInstaller: у командному рядку системи прописуємо «pip install pyinstaller» (рис. 4.6).

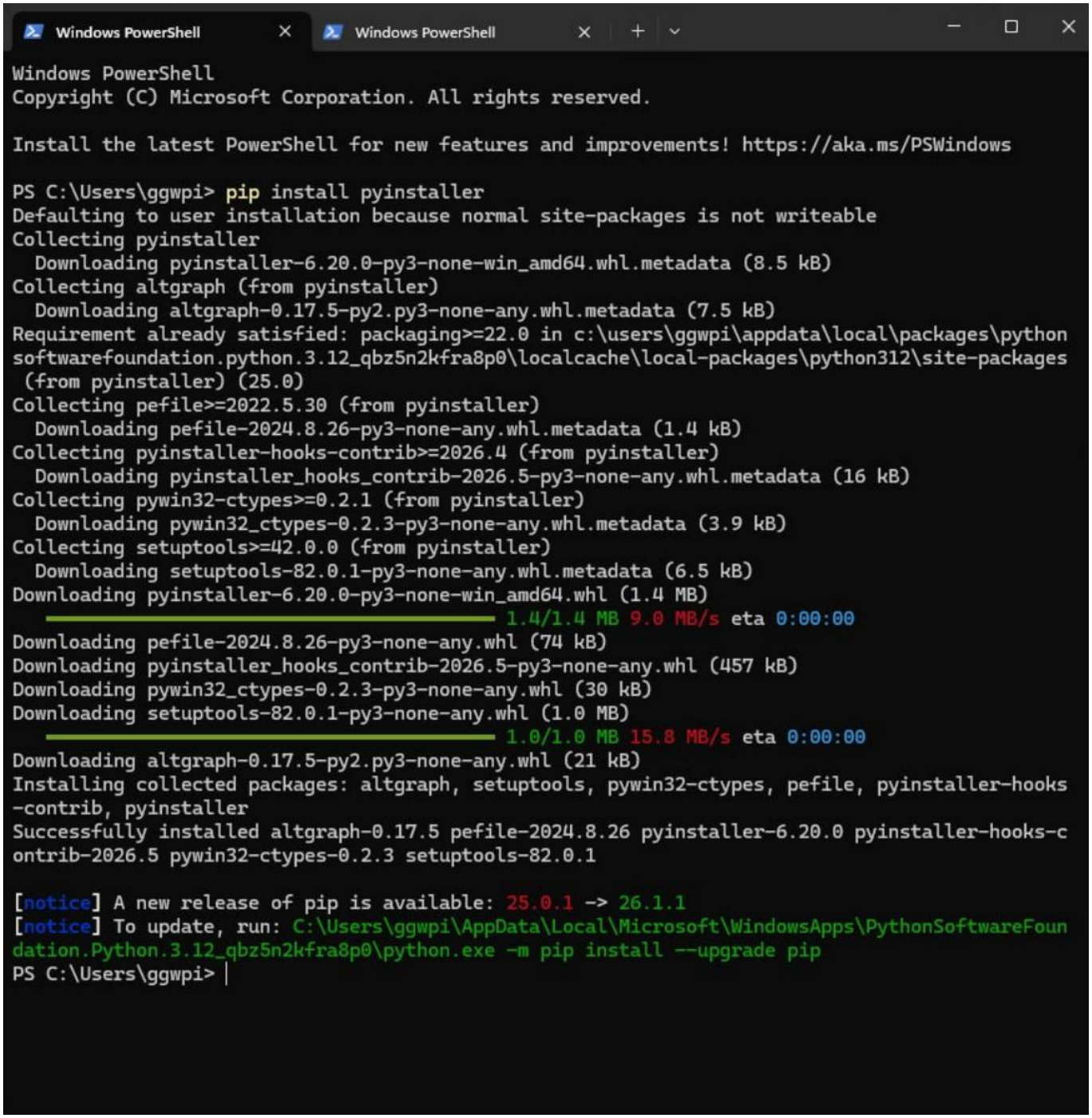

Рисунок 4.6 – Приклад введення запиту до командного рядка Windows

- Підготовка скрипту: переконатися, що основний .py-файл містить усі імпорти та не вимагає додаткових ресурсів із зовнішніх директорій (усе має бути самодостатнім).
- Команда для створення .exe: pyinstaller --onefile --windowed your_script.py (рис. 4.7).

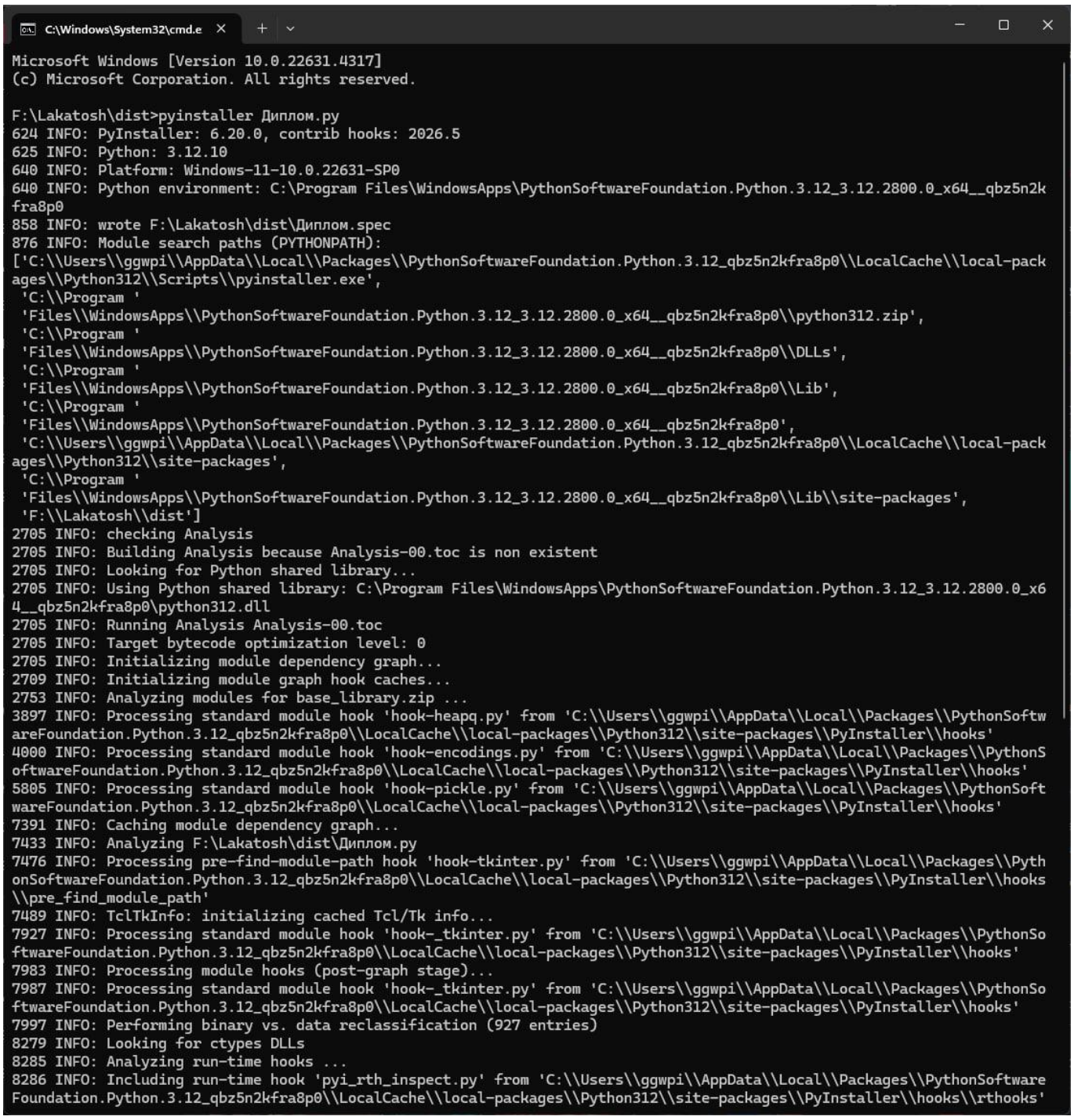

Рисунок 4.7 – Процес створення виконуваного файлу через командний рядок

- --onefile – створює один виконуваний файл (без додаткових папок);
- --windowed – приховує консоль (для GUI-додатків);
- your_script.py – назва основного Python-файлу.
- Готовий файл з'являється у папці dist/ у каталозі проєкту (рис. 4.8).

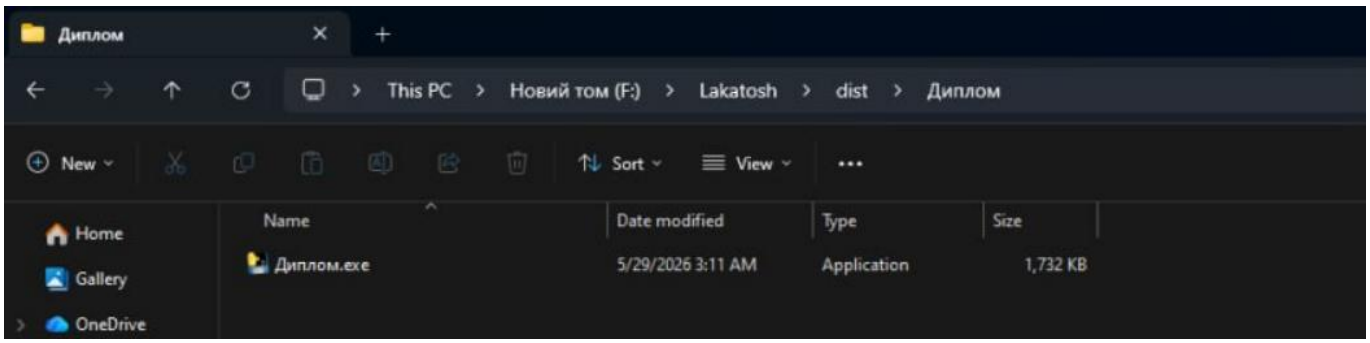

Рисунок 4.8 – Виконуваний файл у папці dist/ у каталозі проєкту dist/Диплом.exe

- Перевірка: виконуваний файл тестується на іншому комп'ютері, щоб переконатися, що він дійсно працює автономно.

Результатом є зручний графічний додаток, що відкривається подвійним кліком та не потребує встановлення Python або додаткових бібліотек. Це дозволяє легко поширювати розроблену систему серед викладачів, студентів або як навчальний інструмент у закритому середовищі.

4.4. Тестування: підхід, результати та приклади

Тестування є невід'ємною частиною розробки програмного забезпечення, особливо у випадку систем, що реалізують складні алгоритми оптимізації. Метою тестування у рамках даної роботи було не лише перевірити технічну коректність реалізованих алгоритмів, а й забезпечити впевненість у стабільності, надійності та передбачуваності поведінки системи у різних умовах використання.

Під час створення системи моделювання та розв'язування транспортних задач використовувались структурні тести, які охоплюють основні обчислювальні сценарії. Основна увага приділялась перевірці коректності перевірки балансу, побудови опорного плану різними методами, оптимізації методом потенціалів, а також правильному формуванню журналу розв'язування. Особливу увагу також приділено взаємодії користувача з інтерфейсом – від уведення вихідних даних до інтерпретації отриманого плану. Простота і наочність відображення проміжних

кроків дозволила протестувати систему не лише на технічному рівні, але й з погляду зручності її використання.

Для тестування було підібрано ряд прикладів різної складності, що дозволило виявити переваги обраної архітектури та перевірити, як система поводиться при роботі як з простими, так і зі складнішими задачами. Було підготовлено набір тестових прикладів, які охоплюють:

- збалансовану задачу малої розмірності;
- незбалансовану (відкриту) задачу;
- задачу з виродженим опорним планом;
- порівняння методів побудови опорного плану на одній задачі.

Приклад 1: збалансована задача

- Запаси: 30, 40; Потреби: 20, 50
- Метод побудови: північно-західного кута
- Журнал розв'язування (рис. 4.9)

Процес тестування суттєво полегшено вбудованою панеллю автоматизації: користувач може обрати один із готових прикладів задач (збалансована 2×2, 3×3, 3×4 або відкрита 2×3) або згенерувати випадкову задачу, не вводячи дані вручну. Окрім того, функція «Порівняти методи» автоматично прогонить задачу всіма трьома методами побудови опорного плану та зведе результати у порівняльну таблицю — це особливо корисно для аналізу різниці між методами на одній і тій самій задачі.

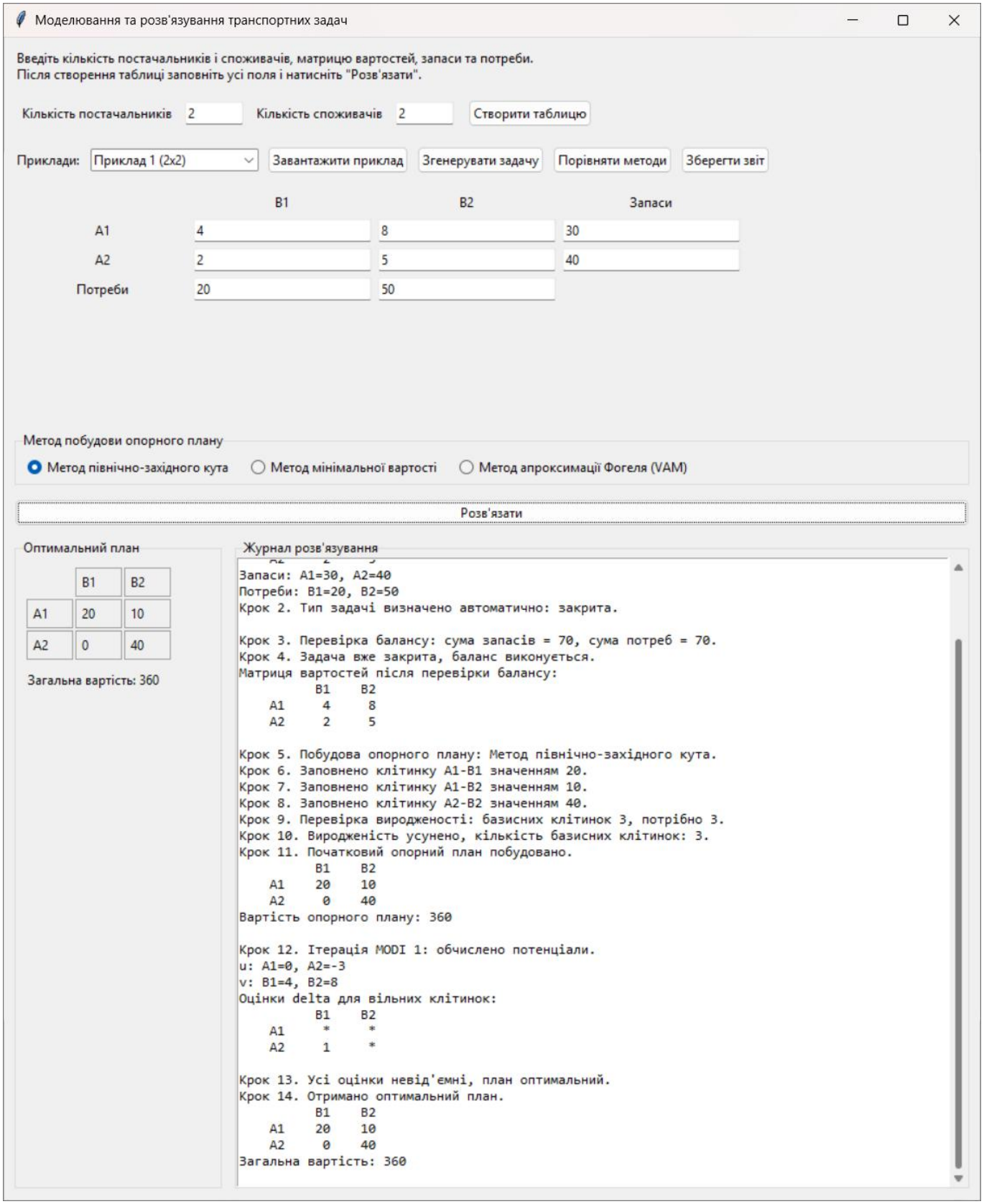

Рисунок 4.9 – Збалансована задача малої розмірності

Приклад 2: незбалансована (відкрита) задача

- Сумарні запаси не дорівнюють сумарним потребам

- Уведення фіктивного споживача
- Журнал розв'язування (рис. 4.10)

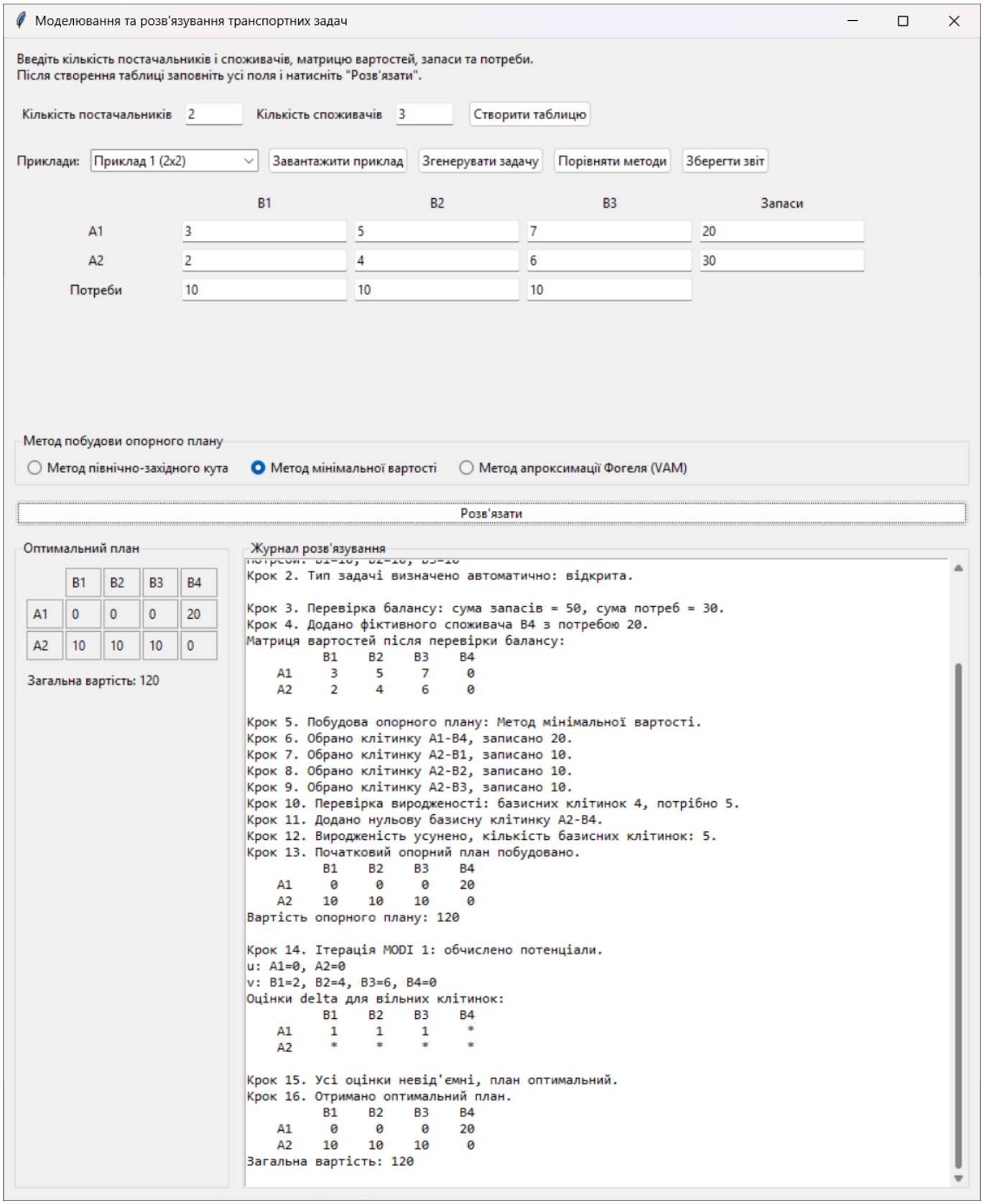

Рисунок 4.10 – Незбалансована задача з фіктивним споживачем

Приклад 3: вироджений опорний план

- Кількість базисних клітин менша за $m + n - 1$
- Уведення нульового базисного перевезення
- Журнал розв'язування (рис. 4.11)

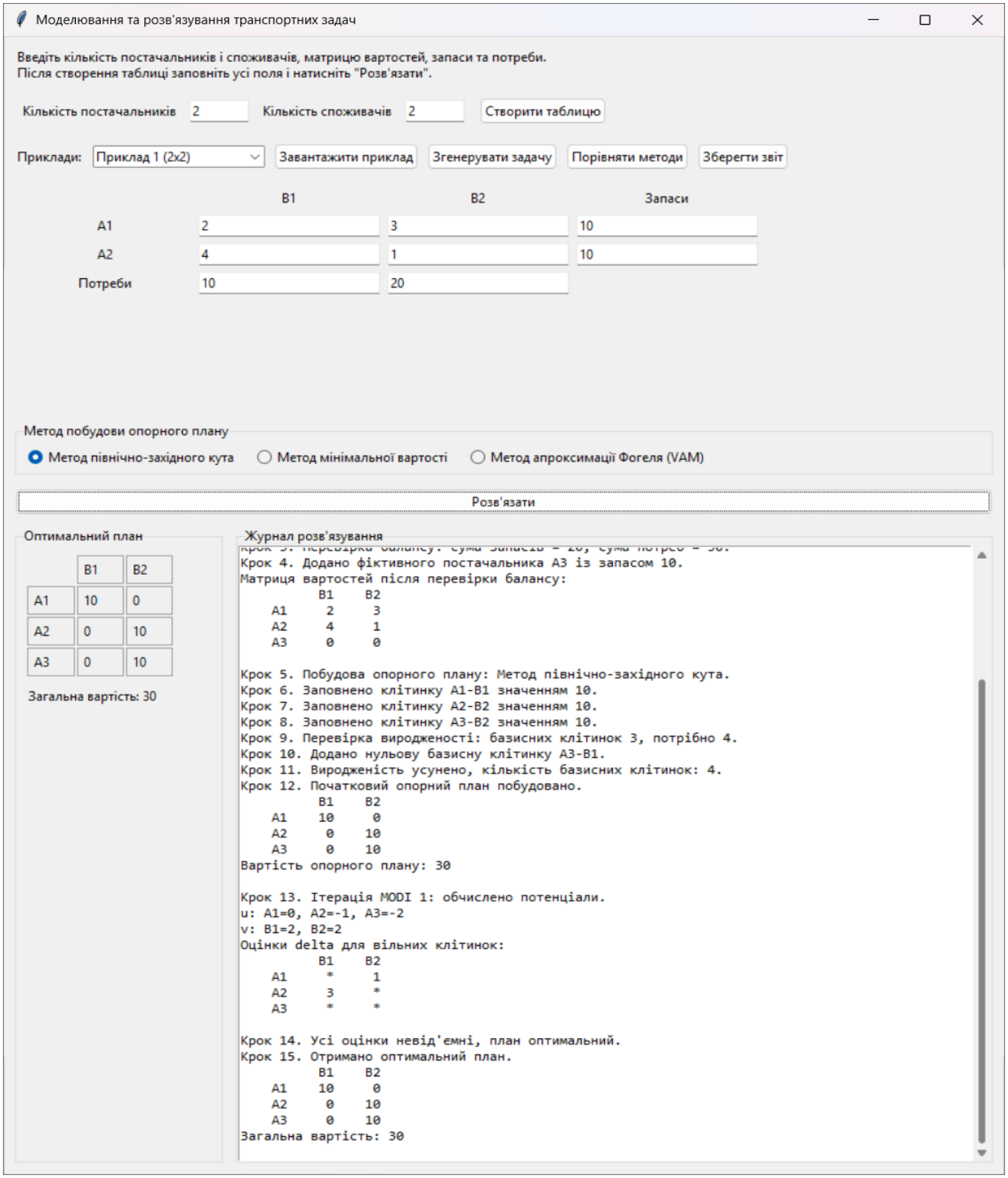

Рисунок 4.11 – Вироджений опорний план

Приклад 4: порівняння методів побудови

- Одна задача розв'язана методами північно-західного кута та апроксимації Фогеля

- Журнал розв'язування (рис. 4.1 1 та рис. 4.12)

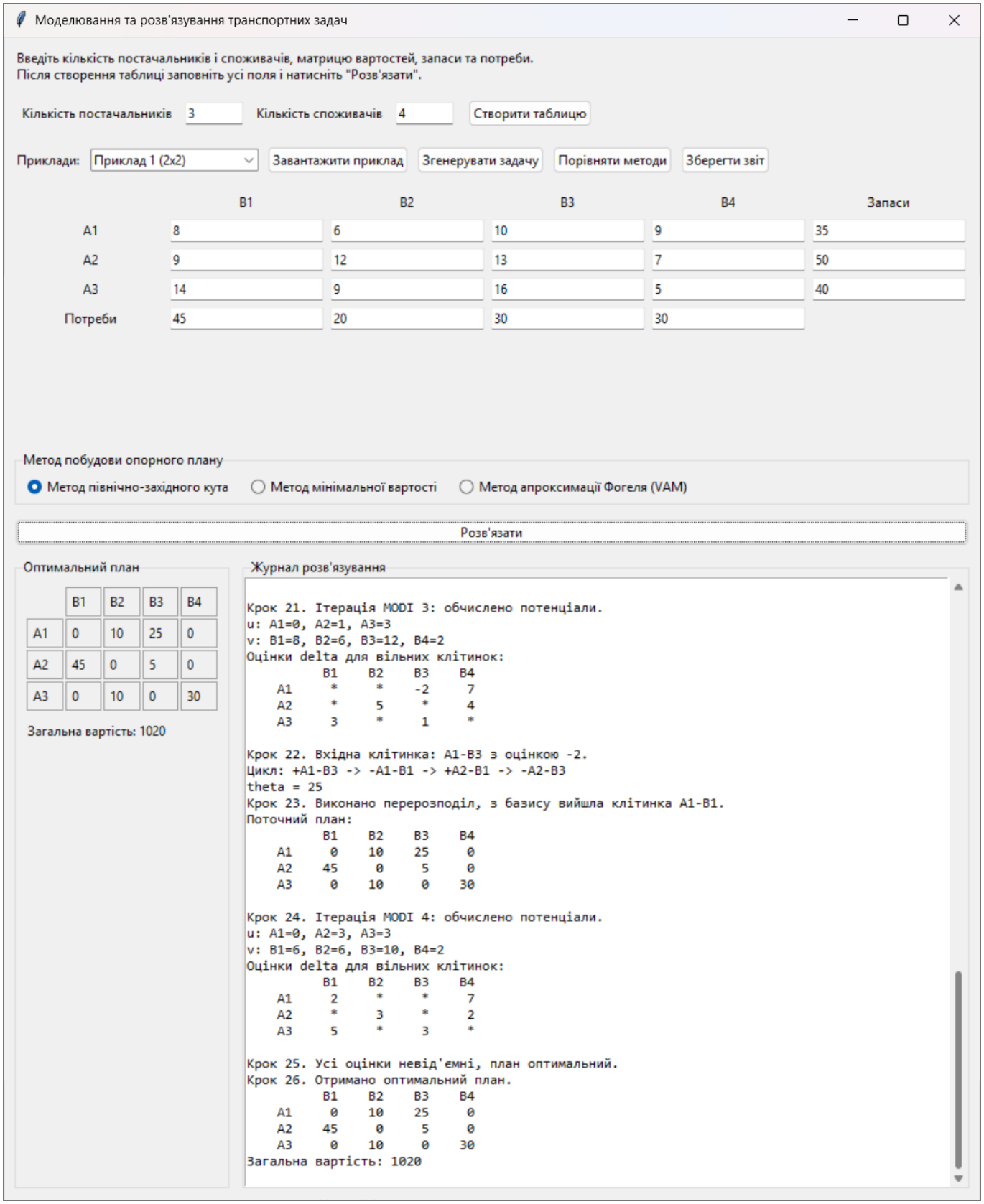

Рисунок 4.12 – Порівняння методів побудови опорного плану - Метод північно-західного кута

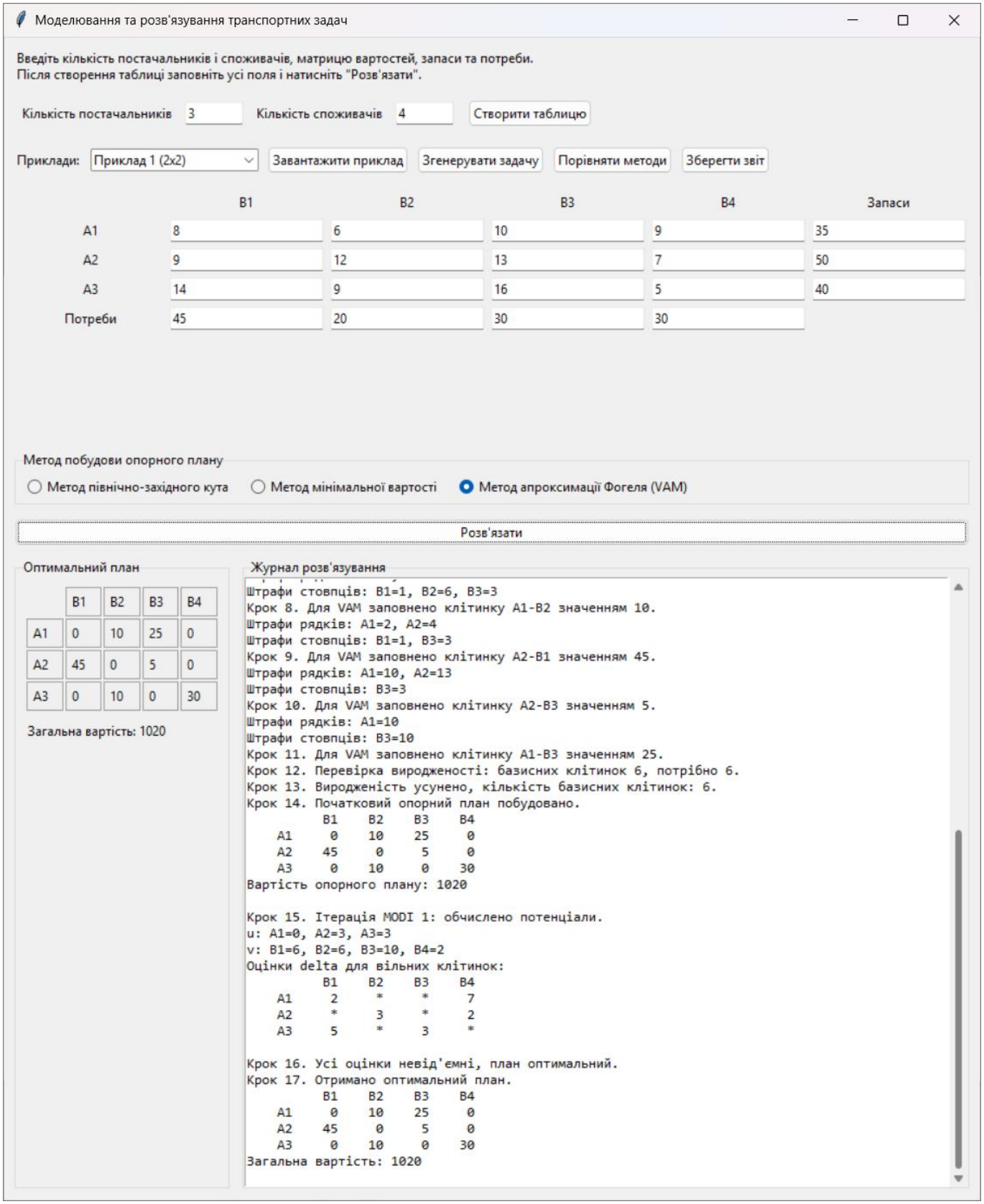

Рисунок 4.13 – Порівняння методів побудови опорного плану - Метод апроксимації Фогеля (VAM)

Ці приклади підтверджують працездатність розробленої системи, коректність реалізації методів побудови опорного плану та оптимізації, а також демонструють зручність інтерфейсу при роботі із задачами такого типу. Система продемонструвала стабільну та передбачувану поведінку навіть за умов збільшення розмірності задачі. Основні обчислювальні операції – побудова опорного плану, обчислення потенціалів та перерозподіл обсягів – виконуються з високою швидкістю. Всі приклади були оброблені миттєво, що вказує на ефективну реалізацію алгоритмів і належну оптимізацію структур даних. Таким чином, результати тестування підтверджують доцільність обраної архітектури, алгоритмічного підходу та практичну придатність системи для навчального і дослідницького використання.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено програму для моделювання та розв'язування транспортних задач. Робота охоплює повний цикл – від постановки задачі до створення працездатного програмного інструменту. У ній поєднано елементи методів дослідження операцій з простим графічним інтерфейсом, який можна використовувати в навчальних цілях або для ознайомлення з принципами оптимізації перевезень.

Окрему увагу приділено методам побудови опорного плану та оптимізації методом потенціалів – фундаментальним підходам до розв'язування транспортної задачі. У роботі їх реалізовано у повному обсязі, з усіма необхідними перевірками та супровідним протоколюванням кроків. Завдяки цьому забезпечується не лише точність обчислень, а й прозорість самого процесу для користувача.

Розроблена система повністю функціональна і підтримує візуалізацію всіх етапів розв'язування – для навчального контексту це принципово. Користувач отримує не лише готовий оптимальний план перевезень, а й бачить шлях, яким цей результат був досягнутий. Такий підхід допомагає формувати аналітичне мислення у студентів та поглиблювати розуміння методів оптимізації – навичок, важливих у дослідженні операцій, лінійному програмуванні й логістиці.

На практиці система показала стабільну та передбачувану роботу. Проведена серія експериментів з різними конфігураціями вхідних даних підтвердила універсальність підходу й надійність обраної архітектури. Продуктивність залишалась на високому рівні навіть для задач більшої розмірності, що свідчить про ефективну реалізацію основних обчислювальних алгоритмів. Крім того, перевірка функціональності в реальному використанні довела, що програмне забезпечення відповідає очікуванням цільової аудиторії.

Поряд із технічними характеристиками, важливою перевагою є зручний графічний інтерфейс, який робить взаємодію з системою доступною навіть для

користувачів без попереднього досвіду роботи з методами оптимізації. Це відкриває простір для використання програми в освітніх установах – на лекціях, семінарах, лабораторних роботах, а також як допоміжного інструмента для викладачів і науковців. У навчальному процесі вона допомагає зробити абстрактні теоретичні поняття наочними, а сам процес навчання – інтерактивнішим і зрозумілішим.

Отримані результати підтверджують, що обрана архітектура, методологія реалізації й програмна платформа є доцільними для розв'язання поставленої задачі. Розроблене програмне забезпечення має простір для розширення, інтеграції в більші системи та використання у прикладних галузях – у логістиці, плануванні перевезень, оптимізації розподілу ресурсів.

У перспективі, з огляду на модульну структуру програми, її можна розширити такими функціями:

- підтримкою додаткових методів оптимізації – наприклад, методу стрибання по клітинах;
- розв'язуванням задач транспортного типу з обмеженнями на пропускну здатність маршрутів;
- інтеграцією з іншими системами планування та логістики;
- веб-версією для онлайн-доступу;
- адаптацією інтерфейсу для мобільних пристроїв;
- імпортом та експортом вихідних даних із зовнішніх файлів (CSV, Excel).

Підсумовуючи, результати кваліфікаційної роботи мають як навчальне, так і прикладне значення і водночас можуть слугувати основою для подальших досліджень у галузі дослідження операцій та оптимізації транспортних процесів. Система відповідає сучасним вимогам до програмних засобів для моделювання перевезень і добре підходить для використання як в академічному, так і в дослідницькому середовищі.

РЕКОМЕНДАЦІЇ

На основі результатів роботи можна сформулювати низку практичних рекомендацій – як щодо використання вже наявної системи, так і щодо її подальшого розвитку.

Щодо застосування у навчальному процесі:

- залучати розроблену систему до викладання дисциплін «Дослідження операцій», «Математичне програмування», «Методи оптимізації» та «Логістика»;
- використовувати програму як демонстраційний інструмент під час лекцій і лабораторних робіт – для наочного пояснення методів побудови опорного плану та оптимізації методом потенціалів;
- пропонувати студентам систему для самостійного опрацювання матеріалу під час підготовки до контрольних і модульних робіт.

Щодо роботи із системою:

- починати ознайомлення з невеликих задач розмірності 2×2 або 3×3, поступово переходячи до складніших прикладів;
- порівнювати між собою опорні плани, побудовані різними методами (північно-західного кута, мінімальної вартості, апроксимації Фогеля), щоб краще зрозуміти переваги кожного підходу;
- уважно опрацьовувати журнал розв'язування – саме він дає розуміння логіки роботи MODI та обґрунтування кожного кроку оптимізації.

Щодо подальшого розвитку системи:

- додати реалізацію методу стрибання по клітинах (stepping-stone) із графічним відображенням замкнених циклів перерозподілу;
- розширити функціонал підтримкою задач транспортного типу з обмеженнями на пропускну здатність маршрутів і задач багатопродуктового перевезення;

- передбачити імпорт та експорт вихідних даних у форматах CSV та Excel для зручної роботи з реальними наборами даних;
- реалізувати графічне подання плану перевезень у вигляді мережевої схеми «постачальник – споживач» поряд із табличним;
- розглянути створення веб-версії додатку для онлайн-доступу та адаптації інтерфейсу для мобільних пристроїв.

Урахування цих рекомендацій зробить розроблену систему ще зручнішим інструментом і розширить сферу її застосування – як у навчальному процесі, так і у прикладному плануванні перевезень.

СПИСОК ЛІТЕРАТУРИ

1. Ольховська О. В.,Черненко О. О. Методичні рекомендації до виконання кваліфікаційної роботи для студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 Комп`ютерні науки ступеня бакалавра: Полтава : ПУЕТ, 2026. – 57 с.
2. Зайченко Ю. П. Дослідження операцій: підручник. 7-ме вид., перероб. і допов. Київ, 2006.
3. Зайченко О. Ю., Зайченко Ю. П. Дослідження операцій. Збірник задач. Київ, 2007.
4. Ульянченко О. В. Дослідження операцій в економіці: підручник для студентів вузів / Харк. нац. аграр. ун-т ім. В. В. Докучаєва. Харків: Гриф, 2002.
5. Дослідження операцій та методи оптимізації в біології та медицині: навчальний посібник / уклад.: О. К. Городецька, К. Х. Зеленський, Є. А. Настенко, В. А. Павлов. Київ: КПІ ім. Ігоря Сікорського. URL: https://ela.kpi.ua/items/9e0c4fc6-2b94-41c1-8cca-450d92f9909d
6. Дослідження операцій: навчальний посібник. Миколаїв: Миколаївський національний аграрний університет. URL: https://dspace.mnau.edu.ua/jspui/bitstream/123456789/9963/1/Doslidzhennia-operatsii-MB-073.pdf
7. Математичні моделі оптимізації: навчальний посібник. Дніпро: Дніпровський державний аграрний університет. URL: https://dspace.dsau.dp.ua/bitstream/123456789/7943/1/%D0%9F%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA_%D0%92%D0%B0%D1%81%D0%B8%D0%BB%D1%8C%D1%94%D0%B2%D0%B0_%D0%9C%D0%BE%D1%80%D0%BE%D0%B7-1.pdf
8. Математичні методи дослідження операцій: навчальний посібник. Львів: Львівський державний університет безпеки життєдіяльності. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/10692/1/%D0%94%D0%9E%D1%8

3%D0%A2%D0%A1_%D0%BF%D1%80%D0%B0%D0%B2%D0%BA%D0%B0%20-%20%D0%BA%D0%BE%D0%BF%D0%B8%D1%8F.pdf

9. Ачкасов А. Є., Воронков О. О. Дослідження операцій: методичні вказівки. Харків: ХНАМГ, 2012. 50 с. URL: https://core.ac.uk/download/pdf/11336808.pdf
10. Транспортна задача. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Транспортна_задача
11. Метод північно-західного кута. MathROS. URL: https://www.mathros.net.ua/metod-pivnichno-zahidnogo-kuta.html
12. Знаходження оптимального плану транспортної задачі методом потенціалів. MathROS. URL: https://www.mathros.net.ua/znahodzhennja-optymalnogo-planu-transportnoi-zadachi-metodom-potencialiv.html
13. Linear programming. Wikipedia. URL: https://en.wikipedia.org/wiki/Linear_programming
14. Simplex algorithm. Wikipedia. URL: https://en.wikipedia.org/wiki/Simplex_algorithm
15. tkinter – Python interface to Tcl/Tk. Python documentation. URL: https://docs.python.org/3/library/tkinter.html
16. PyInstaller Manual. PyInstaller. URL: https://pyinstaller.org/en/stable/
17. The Python Tutorial. Python documentation. URL: https://docs.python.org/3/tutorial/
18. Welcome to Python.org. Python Software Foundation. URL:https://www.python.org/
19. Taha H. A. Operations Research: An Introduction. 10th ed. Pearson, 2017. ISBN 978-0134444017.
20. Hillier F. S., Lieberman G. J. Introduction to Operations Research. 11th ed. New York: McGraw-Hill Education, 2021. ISBN 978-1259872990.
21. Winston W. L. Operations Research: Applications and Algorithms. 4th ed. Belmont: Thomson Brooks/Cole, 2004

ДОДАТОК А. Повний код програми з коментарями

```
 # Приклад для перевірки:
 # Кількість постачальників: 2
 # Кількість споживачів: 2
 # Запаси: A1=30, A2=40
 # Потреби: B1=20, B2=50
 # Матриця вартостей:
 # 4 8
 # 2 5
 import tkinter as tk
 from tkinter import ttk, scrolledtext, messagebox, filedialog
 import copy, random
 root = entry_a = entry_b = table_frame = result_table = total_label = log_box =
method_var = None
 example_box = None
 cost_entries, supply_entries, demand_entries = [], [], []
 step_num = 0
 quiet = False          # коли True - журнал не пишеться (для порівняння методів)
 last_result = None      # збережений останній план для звіту
 # Готові приклади: (матриця вартостей, запаси, потреби)
 EXAMPLES = {
    "Приклад 1 (2x2)": ([[4, 8], [2, 5]], [30, 40], [20, 50]),
    "Приклад 2 (3x3)": ([[2, 7, 4], [3, 3, 1], [5, 4, 6]], [40, 30, 30], [25, 45, 30]),
    "Приклад 3 (3x4)": ([[8, 6, 10, 9], [9, 12, 13, 7], [14, 9, 16, 5]],
                        [50, 60, 50], [30, 40, 50, 40]),
    "Приклад 4 (відкрита 2x3)": ([[3, 5, 7], [6, 4, 2]], [40, 30], [20, 25, 35]),
 }
 def add_step(text):
    global step_num
    if quiet: return
    step_num += 1
    log_box.insert(tk.END, "Крок " + str(step_num) + ". " + text + "\n");
log_box.see(tk.END)
 def add_text(text=""):
    if quiet: return
    log_box.insert(tk.END, text + "\n"); log_box.see(tk.END)
 def clear_log():
    global step_num
    step_num = 0; log_box.delete("1.0", tk.END)
 def make_names(prefix, count): return [prefix + str(i + 1) for i in range(count)]
 def cname(rows, cols, i, j): return rows[i] + "-" + cols[j]
 def get_int(text, name):
    text = text.strip()
    if text == "":
```

```
        raise ValueError("Поле \"" + name + "\" порожнє.")
    try:
        value = int(text)
    except ValueError:
        raise ValueError("Поле \"" + name + "\" має містити ціле невід'ємне число.")
    if value < 0:
        raise ValueError("Поле \"" + name + "\" має містити ціле невід'ємне число.")
    return value
def get_count(text, name):
    value = get_int(text, name)
    if value <= 0: raise ValueError("Поле \"" + name + "\" має бути більшим за нуль.")
    return value
def fmt_matrix(mat, rows, cols):
    vals = rows + cols
    for row in mat:
        vals += [str(x) for x in row]
    w = max(6, max(len(str(x)) for x in vals) + 2)
    s = " " * w + "".join(str(c).rjust(w) for c in cols) + "\n"
    for i in range(len(rows)):
        s += str(rows[i]).rjust(w)
        s += "".join(str(mat[i][j]).rjust(w) for j in range(len(cols))) + "\n"
    return s.rstrip()
def create_table():
    global cost_entries, supply_entries, demand_entries
    try:
        m = get_count(entry_a.get(), "Кількість постачальників")
        n = get_count(entry_b.get(), "Кількість споживачів")
    except ValueError as err:
        messagebox.showerror("Помилка", str(err)); return
    for child in table_frame.winfo_children(): child.destroy()
    cost_entries, supply_entries, demand_entries = [], [], []
    ttk.Label(table_frame, text="").grid(row=0, column=0, padx=3, pady=3)
    for j in range(n):
        ttk.Label(table_frame, text="B" + str(j + 1)).grid(row=0, column=j + 1, padx=3,
pady=3)
    ttk.Label(table_frame, text="Запаси").grid(row=0, column=n + 1, padx=3, pady=3)
    for i in range(m):
        ttk.Label(table_frame, text="A" + str(i + 1)).grid(row=i + 1, column=0, padx=3,
pady=3)
        row = []
        for j in range(n):
            e = ttk.Entry(table_frame, width=9); e.grid(row=i + 1, column=j + 1, padx=3,
pady=3, sticky="ew")
            row.append(e)
        cost_entries.append(row)
```

```
        e = ttk.Entry(table_frame, width=9); e.grid(row=i + 1, column=n + 1, padx=3,
pady=3, sticky="ew")
        supply_entries.append(e)
    ttk.Label(table_frame, text="Потреби").grid(row=m + 1, column=0, padx=3, pady=3)
    for j in range(n):
        e = ttk.Entry(table_frame, width=9); e.grid(row=m + 1, column=j + 1, padx=3,
pady=3, sticky="ew")
        demand_entries.append(e)
    for j in range(n + 2): table_frame.columnconfigure(j, weight=1)
 def fill_entries(costs, supply, demand):
    # заповнити поля готовими даними (для прикладів і генератора)
    entry_a.delete(0, tk.END); entry_a.insert(0, str(len(costs)))
    entry_b.delete(0, tk.END); entry_b.insert(0, str(len(demand)))
    create_table()
    if not cost_entries: return
    for i in range(len(costs)):
        for j in range(len(demand)):
            cost_entries[i][j].insert(0, str(costs[i][j]))
        supply_entries[i].insert(0, str(supply[i]))
    for j in range(len(demand)):
        demand_entries[j].insert(0, str(demand[j]))
 def load_example():
    costs, supply, demand = EXAMPLES[example_box.get()]
    fill_entries(costs, supply, demand)
 def generate_task():
    # випадкова транспортна задача для тестування
    m, n = random.randint(2, 4), random.randint(2, 4)
    costs = [[random.randint(1, 9) for _ in range(n)] for _ in range(m)]
    supply = [random.randint(10, 40) for _ in range(m)]
    demand = [random.randint(10, 40) for _ in range(n)]
    fill_entries(costs, supply, demand)
 def read_data():
    if not cost_entries:
        messagebox.showerror("Помилка", "Спочатку натисніть кнопку \"Створити
таблицю\".")
        return None
    m, n = len(cost_entries), len(cost_entries[0])
    rows, cols = make_names("A", m), make_names("B", n)
    costs, supply, demand = [], [], []
    try:
        for i in range(m):
            row = []
            for j in range(n):
                row.append(get_int(cost_entries[i][j].get(), "вартість " + cname(rows, cols, i,
j)))
```

```
            costs.append(row)
            supply.append(get_int(supply_entries[i].get(), "Запаси " + rows[i]))
        for j in range(n):
            demand.append(get_int(demand_entries[j].get(), "Потреби " + cols[j]))
    except ValueError as err:
        messagebox.showerror("Помилка", str(err)); return None
    return costs, supply, demand, rows, cols
 def balance(costs, supply, demand, rows, cols):
    costs = copy.deepcopy(costs)
    supply, demand, rows, cols = supply[:], demand[:], rows[:], cols[:]
    s1, s2 = sum(supply), sum(demand)
    add_step("Перевірка балансу: сума запасів = " + str(s1) + ", сума потреб = " +
str(s2) + ".")
    if s1 > s2:
        diff = s1 - s2
        for row in costs: row.append(0)
        demand.append(diff)
        cols.append("B" + str(len(cols) + 1))
        add_step("Додано фіктивного споживача " + cols[-1] + " з потребою " + str(diff)
+ ".")
    elif s2 > s1:
        diff = s2 - s1
        costs.append([0 for _ in demand])
        supply.append(diff)
        rows.append("A" + str(len(rows) + 1))
        add_step("Додано фіктивного постачальника " + rows[-1] + " із запасом " +
str(diff) + ".")
    else:
        add_step("Задача вже закрита, баланс виконується.")
    add_text("Матриця вартостей після перевірки балансу:");
add_text(fmt_matrix(costs, rows, cols)); add_text()
    return costs, supply, demand, rows, cols
 def empty_plan(m, n): return [[0 for _ in range(n)] for _ in range(m)]
 def north_west(costs, supply, demand, rows, cols):
    m, n = len(supply), len(demand)
    plan, basis = empty_plan(m, n), set()
    s, d = supply[:], demand[:]
    i, j = 0, 0
    add_step("Побудова опорного плану: Метод північно-західного кута.")
    while i < m and j < n:
        x = min(s[i], d[j])
        plan[i][j] = x
        basis.add((i, j))
        add_step("Заповнено клітинку " + cname(rows, cols, i, j) + " значенням " + str(x)
+ ".")
```

```
            s[i] -= x; d[j] -= x
            if s[i] == 0 and d[j] == 0:
                i += 1; j += 1
            elif s[i] == 0:
                i += 1
            else:
                j += 1
        return plan, basis
    def least_cost(costs, supply, demand, rows, cols):
        m, n = len(supply), len(demand)
        plan, basis = empty_plan(m, n), set()
        s, d = supply[:], demand[:]
        add_step("Побудова опорного плану: Метод мінімальної вартості.")
        while True:
            cells = [(costs[i][j], i, j) for i in range(m) for j in range(n) if s[i] > 0 and d[j] > 0]
            if not cells:
                break
            cells.sort()
            _, i, j = cells[0]
            x = min(s[i], d[j])
            plan[i][j] = x
            basis.add((i, j))
            add_step("Обрано клітинку " + cname(rows, cols, i, j) + ", записано " + str(x) +
"." )
            s[i] -= x; d[j] -= x
        return plan, basis
    def penalty(vals):
        vals = sorted(vals)
        if len(vals) == 1:
            return vals[0]
        return vals[1] - vals[0]
    def vogel(costs, supply, demand, rows, cols):
        m, n = len(supply), len(demand)
        plan, basis = empty_plan(m, n), set()
        s, d = supply[:], demand[:]
        add_step("Побудова опорного плану: Метод апроксимації Фогеля (VAM).")
        while True:
            ar = [i for i in range(m) if s[i] > 0]; ac = [j for j in range(n) if d[j] > 0]
            if not ar or not ac:
                break
            choices, rt, ct = [], [], []
            for i in ar:
                vals = [costs[i][j] for j in ac]
                p = penalty(vals)
                rt.append(rows[i] + "=" + str(p))
```

```
            choices.append(("r", i, p, min(vals)))
        for j in ac:
            vals = [costs[i][j] for i in ar]
            p = penalty(vals)
            ct.append(cols[j] + "=" + str(p))
            choices.append(("c", j, p, min(vals)))
        add_text("Штрафи рядків: " + ", ".join(rt))
        add_text("Штрафи стовпців: " + ", ".join(ct))
        choices.sort(key=lambda x: (-x[2], x[3], x[0], x[1]))
        kind, idx, _, _ = choices[0]
        if kind == "r":
            cells = [(costs[idx][j], -min(s[idx], d[j]), idx, j) for j in ac]
        else:
            cells = [(costs[i][idx], -min(s[i], d[idx]), i, idx) for i in ar]
        cells.sort()
        _, _, i, j = cells[0]
        x = min(s[i], d[j])
        plan[i][j] = x
        basis.add((i, j))
        add_step("Для VAM заповнено клітинку " + cname(rows, cols, i, j) + " значенням
" + str(x) + ".")
        s[i] -= x; d[j] -= x
    return plan, basis
 def has_path(basis, start, finish):
    stack, used = [start], set()
    while stack:
        node = stack.pop()
        if node == finish:
            return True
        if node in used:
            continue
        used.add(node)
        for i, j in basis:
            if node == ("r", i): stack.append(("c", j))
            if node == ("c", j): stack.append(("r", i))
    return False
 def fix_degenerate(plan, basis, costs, rows, cols):
    m, n = len(plan), len(plan[0])
    need = m + n - 1
    add_step("Перевірка виродженості: базисних клітинок " + str(len(basis)) + ",
потрібно " + str(need) + ".")
    while len(basis) < need:
        cells = [(costs[i][j], i, j) for i in range(m) for j in range(n) if (i, j) not in basis and
plan[i][j] == 0]
        cells.sort()
```

```
        ok = False
        for _, i, j in cells:
            if not has_path(basis, ("r", i), ("c", j)):
                basis.add((i, j))
                add_step("Додано нульову базисну клітинку " + cname(rows, cols, i, j) + ".")
                ok = True
                break
        if not ok:
            break
    add_step("Виродженість усунено, кількість базисних клітинок: " + str(len(basis)) + ".")
 def potentials(costs, basis):
    m, n = len(costs), len(costs[0])
    u, v = [None for _ in range(m)], [None for _ in range(n)]
    u[0] = 0
    while True:
        changed = False
        for i, j in basis:
            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
        if all(x is not None for x in u) and all(x is not None for x in v):
            return u, v
        if not changed:
            for i in range(m):
                if u[i] is None:
                    u[i] = 0; changed = True; break
        if not changed:
            for j in range(n):
                if v[j] is None:
                    v[j] = 0; break
 def deltas(costs, basis, u, v):
    m, n = len(costs), len(costs[0])
    tab, best, best_val = [["*" for _ in range(n)] for _ in range(m)], None, 0
    for i in range(m):
        for j in range(n):
            if (i, j) not in basis:
                val = costs[i][j] - (u[i] + v[j])
                tab[i][j] = val
                if val < best_val:
                    best, best_val = (i, j), val
    return tab, best, best_val
 def find_cycle(start, basis, m, n):
    allowed = set(basis)
```

```
    allowed.add(start)
    def go(cell, path, horizontal):
        i, j = cell
        if horizontal:
            cells = [(i, col) for col in range(n) if col != j and (i, col) in allowed]
        else:
            cells = [(row, j) for row in range(m) if row != i and (row, j) in allowed]
        for c in cells:
            if c == start and len(path) >= 4:
                return path
            if c not in path:
                res = go(c, path + [c], not horizontal)
                if res: return res
        return None
    res = go(start, [start], True)
    if res: return res
    return go(start, [start], False)
 def show_cycle(cycle, rows, cols):
    parts = []
    for k, (i, j) in enumerate(cycle):
        parts.append(("+" if k % 2 == 0 else "-") + cname(rows, cols, i, j))
    return " -> ".join(parts)
 def optimize(costs, plan, basis, rows, cols):
    m, n, it = len(costs), len(costs[0]), 1
    while True:
        u, v = potentials(costs, basis)
        add_step("Ітерація MODI " + str(it) + ": обчислено потенціали.")
        add_text("u: " + ", ".join(rows[i] + "=" + str(u[i]) for i in range(m)))
        add_text("v: " + ", ".join(cols[j] + "=" + str(v[j]) for j in range(n)))
        tab, enter, best_val = deltas(costs, basis, u, v)
        add_text("Оцінки delta для вільних клітинок:"); add_text(fmt_matrix(tab, rows,
cols)); add_text()
        if enter is None:
            add_step("Усі оцінки невід'ємні, план оптимальний.")
            return plan, basis
        add_step("Вхідна клітинка: " + cname(rows, cols, enter[0], enter[1]) + " з оцінкою
" + str(best_val) + ".")
        cycle = find_cycle(enter, basis, m, n)
        if not cycle:
            add_step("Не вдалося побудувати замкнений цикл для перерозподілу.")
            return plan, basis
        minus = [cycle[k] for k in range(1, len(cycle), 2)]
        theta = min(plan[i][j] for i, j in minus)
        leave = minus[0]
        for c in minus:
```

```
            if plan[c[0]][c[1]] == theta:
                leave = c
                break
        add_text("Цикл: " + show_cycle(cycle, rows, cols)); add_text("theta = " + str(theta))
        basis.add(enter)
        for k, (i, j) in enumerate(cycle):
            plan[i][j] += theta if k % 2 == 0 else -theta
        basis.remove(leave)
        add_step("Виконано перерозподіл, з базису вийшла клітинка " + cname(rows,
cols, leave[0], leave[1]) + ".")
        add_text("Поточний план:"); add_text(fmt_matrix(plan, rows, cols)); add_text()
        it += 1
        if it > 100:
            add_step("Зупинка після 100 ітерацій.")
            return plan, basis
 def get_total(costs, plan):
    total = 0
    for i in range(len(plan)):
        for j in range(len(plan[0])):
            total += costs[i][j] * plan[i][j]
    return total
 def run_solver(method, data):
    # повний цикл розв'язування для обраного методу, повертає план і вартості
    costs, supply, demand, rows, cols = data
    costs, supply, demand, rows, cols = balance(costs, supply, demand, rows, cols)
    if method == "nw":
        plan, basis = north_west(costs, supply, demand, rows, cols)
    elif method == "least":
        plan, basis = least_cost(costs, supply, demand, rows, cols)
    else:
        plan, basis = vogel(costs, supply, demand, rows, cols)
    fix_degenerate(plan, basis, costs, rows, cols)
    init_total = get_total(costs, plan)
    add_step("Початковий опорний план побудовано.")
    add_text(fmt_matrix(plan, rows, cols)); add_text("Вартість опорного плану: " +
str(init_total)); add_text()
    plan, basis = optimize(costs, plan, basis, rows, cols)
    total = get_total(costs, plan)
    return plan, rows, cols, total, init_total
 def draw_result(plan, rows, cols, total):
    for child in result_table.winfo_children(): child.destroy()
    ttk.Label(result_table, text="").grid(row=0, column=0, padx=1, pady=1,
sticky="nsew")
    for j, col in enumerate(cols):
```

```
        ttk.Label(result_table, text=col, relief="solid", padding=4).grid(row=0, column=j +
1, padx=1, pady=1, sticky="nsew")
    for i, row in enumerate(rows):
        ttk.Label(result_table, text=row, relief="solid", padding=4).grid(row=i + 1,
column=0, padx=1, pady=1, sticky="nsew")
        for j in range(len(cols)):
            ttk.Label(result_table, text=str(plan[i][j]), relief="solid", padding=4).grid(row=i +
1, column=j + 1, padx=1, pady=1, sticky="nsew")
    for j in range(len(cols) + 1): result_table.columnconfigure(j, weight=1)
    total_label.config(text="Загальна вартість: " + str(total))
 def solve():
    global last_result
    data = read_data()
    if data is None:
        return
    clear_log()
    costs, supply, demand, rows, cols = data
    add_step("Дані зчитано і перевірено.")
    add_text("Початкова матриця вартостей:"); add_text(fmt_matrix(costs, rows, cols))
    add_text("Запаси: " + ", ".join(rows[i] + "=" + str(supply[i]) for i in range(len(rows))))
    add_text("Потреби: " + ", ".join(cols[j] + "=" + str(demand[j]) for j in
range(len(cols))))
    # автоматичне визначення типу задачі
    typ = "закрита" if sum(supply) == sum(demand) else "відкрита"
    add_step("Тип задачі визначено автоматично: " + typ + ".")
    add_text()
    plan, rows2, cols2, total, _ = run_solver(method_var.get(), data)
    draw_result(plan, rows2, cols2, total)
    last_result = (plan, rows2, cols2, total)
    add_step("Отримано оптимальний план.")
    add_text(fmt_matrix(plan, rows2, cols2)); add_text("Загальна вартість: " + str(total))
 def fmt_compare(summary):
    s = "Метод".ljust(28) + "Опорний".rjust(10) + "Оптимум".rjust(10) + "\n"
    for label, init, total in summary:
        s += label.ljust(28) + str(init).rjust(10) + str(total).rjust(10) + "\n"
    return s.rstrip()
 def compare_methods():
    global quiet
    data = read_data()
    if data is None:
        return
    clear_log()
    add_step("Порівняння методів побудови опорного плану.")
    summary = []
    for m, label in (("nw", "Північно-західного кута"), ("least", "Мінімальної вартості"),
```

```
                ("vam", "Фогеля (VAM)")):
            quiet = True
            plan, r, c, total, init = run_solver(m, data)
            quiet = False
            summary.append((label, init, total))
        add_text(fmt_compare(summary)); add_text()
        best = min(summary, key=lambda x: x[1])
        add_step("Найдешевший опорний план дає метод: " + best[0] + " (вартість " +
str(best[1]) + ").")
        add_text("Оптимальна вартість однакова для всіх методів: " + str(summary[0][2]) +
".")
    def save_report():
        # формування текстового звіту з вихідними даними, журналом і результатом
        if last_result is None:
            messagebox.showinfo("Звіт", "Спочатку розв'яжіть задачу."); return
        path = filedialog.asksaveasfilename(defaultextension=".txt",
                                            filetypes=[("Текстовий файл", "*.txt")], title="Зберегти звіт")
        if not path:
            return
        plan, rows, cols, total = last_result
        try:
            with open(path, "w", encoding="utf-8") as f:
                f.write("Звіт з розв'язування транспортної задачі\n")
                f.write("=" * 42 + "\n\n")
                f.write("Оптимальний план:\n" + fmt_matrix(plan, rows, cols) + "\n\n")
                f.write("Загальна вартість: " + str(total) + "\n\n")
                f.write("Журнал розв'язування:\n")
                f.write(log_box.get("1.0", tk.END))
        except OSError as err:
            messagebox.showerror("Помилка", "Не вдалося зберегти звіт.\n" + str(err)); return
        messagebox.showinfo("Звіт", "Звіт збережено:\n" + path)
    def build_gui():
        global root, entry_a, entry_b, table_frame, result_table, total_label, log_box,
method_var, example_box
        root = tk.Tk(); root.title("Моделювання та розв'язування транспортних задач")
        root.geometry("1050x760"); root.minsize(900, 640)
        root.columnconfigure(0, weight=1)
        for r, w in ((3, 1), (6, 2)):
            root.rowconfigure(r, weight=w)
        text = ("Введіть кількість постачальників і споживачів, матрицю вартостей, "
                "запаси та потреби.\nПісля створення таблиці заповніть усі поля і натисніть
\"Розв'язати\".")
        ttk.Label(root, text=text, justify="left").grid(row=0, column=0, padx=10, pady=(10, 5),
sticky="ew")
        top = ttk.Frame(root); top.grid(row=1, column=0, padx=10, pady=5, sticky="ew")
```

```
    top.columnconfigure(5, weight=1)
    ttk.Label(top, text="Кількість постачальників").grid(row=0, column=0, padx=5,
pady=4)
    entry_a = ttk.Entry(top, width=8); entry_a.grid(row=0, column=1, padx=5, pady=4)
    ttk.Label(top, text="Кількість споживачів").grid(row=0, column=2, padx=5, pady=4)
    entry_b = ttk.Entry(top, width=8); entry_b.grid(row=0, column=3, padx=5, pady=4)
    ttk.Button(top, text="Створити таблицю", command=create_table).grid(row=0,
column=4, padx=8, pady=4)
    # панель автоматизації: приклади, генератор, порівняння, звіт
    tools = ttk.Frame(root); tools.grid(row=2, column=0, padx=10, pady=5, sticky="ew")
    ttk.Label(tools, text="Приклади:").grid(row=0, column=0, padx=(0, 4), pady=4)
    example_box = ttk.Combobox(tools, values=list(EXAMPLES.keys()),
state="readonly", width=22)
    example_box.grid(row=0, column=1, padx=4, pady=4); example_box.current(0)
    ttk.Button(tools, text="Завантажити приклад", command=load_example).grid(row=0,
column=2, padx=4, pady=4)
    ttk.Button(tools, text="Згенерувати задачу", command=generate_task).grid(row=0,
column=3, padx=4, pady=4)
    ttk.Button(tools, text="Порівняти методи",
command=compare_methods).grid(row=0, column=4, padx=4, pady=4)
    ttk.Button(tools, text="Зберегти звіт", command=save_report).grid(row=0, column=5,
padx=4, pady=4)
    table_frame = ttk.Frame(root); table_frame.grid(row=3, column=0, padx=10, pady=5,
sticky="nsew")
    method_var = tk.StringVar(value="nw")
    methods = ttk.LabelFrame(root, text="Метод побудови опорного плану");
methods.grid(row=4, column=0, padx=10, pady=5, sticky="ew")
    radios = [("Метод північно-західного кута", "nw"), ("Метод мінімальної вартості",
"least"), ("Метод апроксимації Фогеля (VAM)", "vam")]
    for c, item in enumerate(radios):
        ttk.Radiobutton(methods, text=item[0], variable=method_var,
value=item[1]).grid(row=0, column=c, padx=8, pady=5, sticky="w")
    ttk.Button(root, text="Розв'язати", command=solve).grid(row=5, column=0, padx=10,
pady=7, sticky="ew")
    bottom = ttk.Frame(root); bottom.grid(row=6, column=0, padx=10, pady=(5, 10),
sticky="nsew")
    for c, w in ((0, 1), (1, 2)):
        bottom.columnconfigure(c, weight=w)
    bottom.rowconfigure(0, weight=1)
    res = ttk.LabelFrame(bottom, text="Оптимальний план"); res.grid(row=0, column=0,
padx=(0, 6), sticky="nsew")
    res.columnconfigure(0, weight=1)
    result_table = ttk.Frame(res); result_table.grid(row=0, column=0, padx=8, pady=8,
sticky="nsew")
```

```
    total_label = ttk.Label(res, text="Загальна вартість: "); total_label.grid(row=1,
column=0, padx=8, pady=(0, 8), sticky="w")
    logf = ttk.LabelFrame(bottom, text="Журнал розв'язування"); logf.grid(row=0,
column=1, padx=(6, 0), sticky="nsew")
    logf.columnconfigure(0, weight=1); logf.rowconfigure(0, weight=1)
    log_box = scrolledtext.ScrolledText(logf, wrap=tk.NONE, font=("Consolas", 10));
log_box.grid(row=0, column=0, sticky="nsew")
    entry_a.insert(0, "2"); entry_b.insert(0, "2")
    return root
 if __name__ == "__main__":
    app = build_gui()
    app.mainloop()
```

ДОДАТОК Б. Інструкція користувача

Запуск програми

1. Якщо ви використовуєте виконуваний файл (.exe), запустіть програму подвійним кліком по ярлику Диплом.exe.
2. Якщо ви запускаєте програму з Python-коду:

- переконайтесь, що у вас встановлений Python 3.7 або новіший;
- бібліотека tkinter входить до стандартної поставки Python, додаткове встановлення не потрібне;
- запустіть файл Диплом.py через термінал або IDE (IDLE, PyCharm, VS Code): python Диплом.py

Правила введення даних

- Кількість постачальників і споживачів – цілі додатні числа.
- Усі значення вартостей перевезень, запасів та потреб – цілі невід'ємні числа.
- Спочатку задається розмірність задачі, потім натискається «Створити таблицю» для формування сітки введення.
- Якщо потрібно змінити розмірність, задайте нові значення кількості та знову натисніть «Створити таблицю».
- Не обов'язково, щоб сума запасів дорівнювала сумі потреб – програма автоматично зведе відкриту задачу до закритого вигляду.

Інтерфейс програми

- Інструкція – у верхній частині вікна наведено короткі правила введення.
- Поля «Кількість постачальників» і «Кількість споживачів» із кнопкою «Створити таблицю».
- Таблиця вартостей з позначенням рядків (A1, A2, …), стовпців (B1, B2, …), стовпця «Запаси» та рядка «Потреби».

- Перемикач методу побудови опорного плану: «Метод північно-західного кута», «Метод мінімальної вартості», «Метод апроксимації Фогеля (VAM)».
- Панель автоматизації – випадальний список «Приклади» з готовими задачами та кнопки: «Завантажити приклад» (заповнює таблицю обраним прикладом), «Згенерувати задачу» (створює випадкову транспортну задачу), «Порівняти методи» (запускає всі три методи на тих самих даних із виведенням порівняльної таблиці), «Зберегти звіт» (експортує результат у текстовий файл).
- Кнопка «Розв'язати» – запускає процес розв'язування.
- Панель «Оптимальний план» – матрична таблиця з підсумковими обсягами перевезень.
- Поле «Загальна вартість» – сумарна вартість оптимального плану.
- Журнал розв'язування – текстова область з повним протоколом кроків побудови та оптимізації.

Покрокова інструкція користування

1. Запустіть програму.
2. Уведіть кількість постачальників і споживачів та натисніть «Створити таблицю».
3. Заповніть матрицю вартостей перевезень.
4. Заповніть стовпець «Запаси» і рядок «Потреби».
5. Оберіть метод побудови опорного плану.
6. Натисніть «Розв'язати».
7. Перегляньте оптимальний план, загальну вартість та повний хід обчислень у журналі.

Додаткові можливості:

– Щоб швидко спробувати програму, оберіть зі списку «Приклади» одну з готових задач і натисніть «Завантажити приклад» – усі поля заповняться автоматично.

– Натисніть «Згенерувати задачу» для створення випадкової задачі з невеликими розмірностями (2×2 – 4×4).

– Кнопка «Порівняти методи» дозволяє побачити, як три методи побудови опорного плану працюють на одній задачі: у журналі з'являється підсумкова таблиця, де для кожного методу зазначено вартість опорного плану та оптимальну вартість.

– Після розв'язування задачі натисніть «Зберегти звіт», щоб вивантажити оптимальний план, загальну вартість і журнал у текстовий файл (.txt) для подальшого аналізу.

Поради та зауваження

- Якщо сума запасів і потреб не збігається, програма автоматично додає фіктивного постачальника або споживача з нульовими вартостями.
- У разі виродженого опорного плану до базису додається нульова базисна клітинка.
- Усі поля повинні містити цілі невід'ємні числа; інакше програма покаже повідомлення про помилку.
- Метод апроксимації Фогеля (VAM) зазвичай дає опорний план, ближчий до оптимального, тому потребує менше ітерацій оптимізації.

Приклад використання

- Постачальники: A1=30, A2=40.
- Споживачі: B1=20, B2=50.
- Матриця вартостей: 4 8 / 2 5.
- Метод побудови: північно-західного кута.
- Очікуваний результат: оптимальний план із загальною вартістю 360.

- Пояснення: програма будує опорний план методом північно-західного кута, обчислює потенціали та оцінки вільних клітинок, виявляє клітинку з від'ємною оцінкою і виконує перерозподіл уздовж замкненого циклу до отримання оптимального плану.

Скріншот інтерфейсу (рис. Б.1)

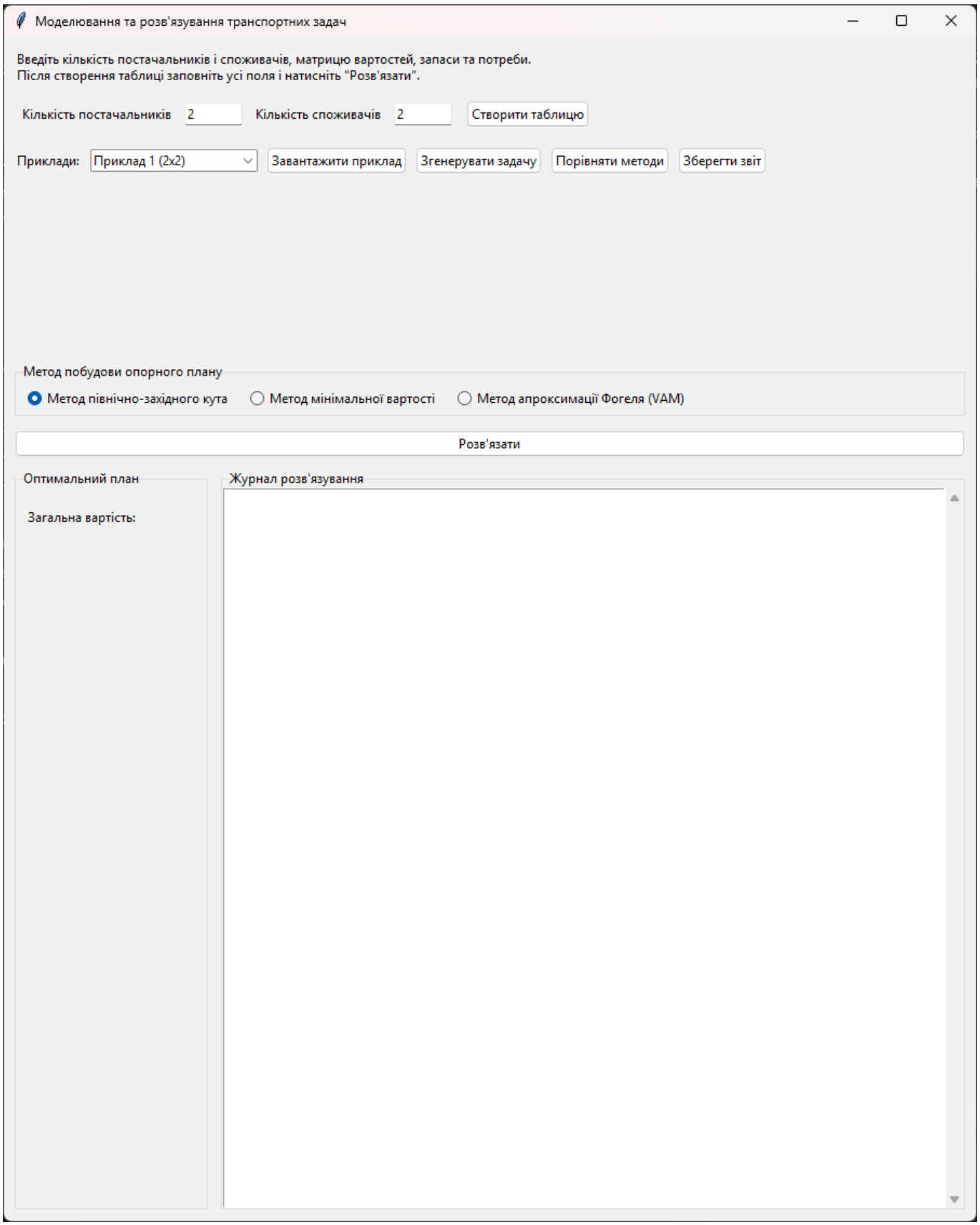

Рисунок Б.1 – Інтерфейс додатку

Програма орієнтована на навчальне використання та забезпечує прозору візуалізацію процесу побудови й оптимізації плану перевезень.