Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

« Програмна реалізація розв'язання логістичних задач на основі генетичного алгоритму »

Виконав ст. гр. КН-22-2 _____ Салоїд М. В.

Керівник _____ Маринич І. А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2026

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *26* » *січня* 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-22-2 *Салоїд Максиму Вікторовичу*

1. Тема кваліфікаційної роботи: *«Програмна реалізація розв'язання логістичних задач на основі генетичного алгоритму»*

затверджено наказом по університету № *173с* від *30.03.2026 р.*

2. Термін здачі кваліфікаційної роботи: *10.06.2026 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка., додатки, презентація у Microsoft PowerPoint в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Маринич І. А.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.26*
2	*Розділ 1*	*05.04.26*
3	*Розділ 2*	*01.05.26*
4	*Висновки*	*25.05.26*
5	*Оформлення кваліфікаційної роботи*	*28.05.26*
6	*Підготовка презентації та графічного матеріалу*	*20.05.26*
7	*Підготовка доповіді до захисту*	*05.06.26*

6. Дата видачі завдання: *28.01.2026р.*

Керівник _____ / Маринич І. А./

7. Запевнення: *Я, Салоїд Максим Вікторович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Салоїд М. В./

АНОТАЦІЯ

Салоїд М. В. Програмна реалізація розв'язання логістичних задач на основі генетичного алгоритму : кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2026. 62 с.

Актуальність теми полягає в тому, що задачі оптимізації логістичних процесів, зокрема транспортні задачі, є важливим елементом ефективного функціонування сучасних підприємств. У реальних умовах такі задачі часто є незбалансованими та багатовимірними, що ускладнює їх розв’язання класичними методами та обумовлює необхідність використання сучасних підходів, зокрема генетичних алгоритмів.

Метою роботи є розробка та дослідження методу розв’язання незбалансованих транспортних задач довільної розмірності на основі генетичного алгоритму, а також створення програмного засобу для їх ефективного розв’язання.

У першому розділі розглянуто теоретичні основи транспортних задач, виконано аналіз класичних методів їх розв’язання та сучасних метаевристичних підходів. Особливу увагу приділено генетичним алгоритмам як ефективному інструменту оптимізації складних задач. Описано принципи побудови генетичного алгоритму, реалізовано острівну модель та підходи до формування початкової популяції.

У другому розділі представлено архітектуру програмної системи, описано особливості реалізації, графічний інтерфейс користувача та функціональні можливості розробленого програмного продукту. Наведено результати тестування, виконано порівняння з класичними методами (метод північно-західного кута, метод мінімального елемента) та розв’язувачем HiGHS, а також представлено візуалізацію отриманих результатів.

Ключові слова:

ТРАНСПОРТНА ЗАДАЧА, ГЕНЕТИЧНИЙ АЛГОРИТМ, ОПТИМІЗАЦІЯ, ЛОГІСТИКА, МЕТАЕВРИСТИКА, ОСТРІВНА МОДЕЛЬ, ПРОГРАМНА РЕАЛІЗАЦІЯ

ЗМІСТ

ВСТУП

Оптимізація розподілу ресурсів є однією з ключових задач прикладної математики та економіки. Одним із найбільш відомих і широко застосовуваних її часткових випадків є транспортна задача, що описує процес мінімізації витрат на доставку продукції від постачальників до споживачів за заданих обсягів постачання та попиту. Класична постановка цієї задачі передбачає баланс між сумарними обсягами пропозиції та попиту. Однак у реальних прикладних умовах, зокрема у сфері логістики, торгівлі та управління ланцюгами постачання, це припущення виконується далеко не завжди.

На практиці значно частіше виникають незбалансовані транспортні задачі, у яких сумарний обсяг постачання не дорівнює сумарному попиту. Це може бути зумовлено виробничими витратами, формуванням резервів, коливаннями попиту або впливом зовнішніх факторів. Розв’язання таких задач потребує адаптації класичних алгоритмів або застосування універсальних підходів, здатних ефективно функціонувати в умовах асиметричних обмежень.

Подальше ускладнення логістичних систем обумовлює необхідність врахування додаткових параметрів, таких як наявність проміжних складів, часові обмеження, різні типи вантажів та інші фактори. Це призводить до формування багатоіндексних (багатовимірних) узагальнень транспортної задачі, у яких кількість індексів перевищує два, а розмірність простору рішень суттєво зростає. За таких умов традиційні методи оптимізації стають обчислювально затратними та обмежено придатними для застосування у складних прикладних сценаріях.

У зв’язку з цим особливої актуальності набуває використання метаевристичних підходів, зокрема генетичних алгоритмів, які здатні ефективно працювати зі складними, слабо формалізованими задачами, що характеризуються великою кількістю обмежень та складною структурою простору рішень. Такі алгоритми можуть бути адаптовані до різних умов, включаючи незбалансовані задачі та задачі довільної розмірності.

У даній кваліфікаційній роботі розглядаються методи розв’язання як класичної, так і багатоіндексної транспортної задачі з акцентом на незбалансовані постановки, які найбільш повно відображають реальні економічні процеси. Передбачено розроблення універсального програмного рішення з графічним інтерфейсом користувача, засобами візуалізації результатів і можливістю їх експорту. Основу запропонованого підходу становить генетичний алгоритм, доповнений механізмами формування початкової популяції з використанням класичних методів, зокрема методу північно-західного кута, методу мінімального елемента, випадкової ініціалізації, а також інтеграцією з промисловим розв’язувачем лінійних задач HiGHS.

Об’єктом дослідження є процеси транспортної оптимізації в логістичних системах.

Предметом дослідження є методи та алгоритми розв’язання незбалансованих транспортних задач із використанням генетичних алгоритмів.

Метою роботи є розробка та дослідження гнучкого методу розв’язання незбалансованих транспортних задач довільної розмірності, здатного ефективно враховувати обмеження та складність реальних логістичних процесів.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати існуючі підходи до розв’язання багатоіндексної транспортної задачі та методи оптимізації;
- розробити універсальне представлення вхідних даних для задач довільної розмірності;
- реалізувати генетичний алгоритм з острівною моделлю та багаторівневою ініціалізацією популяції;
- розробити графічний інтерфейс користувача для постановки задач, відображення результатів та їх експорту;
- провести порівняльний аналіз запропонованого методу з класичними алгоритмами (метод північно-західного кута, метод мінімального елемента) та розв’язувачем HiGHS.

.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ТА АЛГОРИТМІВ ОПТИМІЗАЦІЇ ЛОГІСТИЧНИХ ЗАДАЧ

1.1 Логістичні завдання та їхні особливості

Логістика - це міждисциплінарна галузь, орієнтована на управління матеріальними, інформаційними та фінансовими потоками з метою забезпечення ефективного розподілу ресурсів. Головне завдання логістики - оптимізувати всі етапи руху товарів і послуг - від виробництва до кінцевого споживача. Розв'язання проблем, що виникають у цій сфері, вимагає застосування методів математичного моделювання та аналізу, що пояснюється складністю та універсальністю логістичних процесів.

З точки зору математичного моделювання, логістичні задачі можна класифікувати на кілька основних категорій[1]:

– завдання з управління запасами, спрямовані на оптимізацію обсягів зберігання з урахуванням динаміки попиту, логістичних витрат, часу доставки тощо;

– завдання складської логістики, включаючи розподіл товарів у складських комплексах, а також мінімізацію часових і фінансових витрат на обробку вантажів;

– завдання транспортної логістики, пов'язані з плануванням і організацією перевезення вантажів або пасажирів між пунктами призначення.

Незважаючи на різноманіття логістичних завдань, багато з них зводяться до однієї ключової проблеми - оптимізації транспортних процесів, оскільки транспортний компонент є визначальним фактором ефективності логістичних операцій. Частка транспортних витрат у загальній структурі логістичних витрат може бути значною, що вимагає розробки та впровадження методів оптимізації для зниження витрат з урахуванням існуючих обмежень (місткість транспортних засобів, часові обмеження, маршрутизація).

Транспортні задачі, як особливі випадки логістичних задач, набули статусу

незалежного об'єкта дослідження завдяки своїй практичній значущості та суворій математичній формалізації. Останнє дозволяє ефективно застосовувати методи математичного програмування та алгоритми оптимізації, що робить транспортні задачі зручними для аналізу та моделювання. Розроблені підходи для їх розв'язання є високоадаптивними і застосовуються не лише в галузі транспортної логістики, а й у суміжних сферах, зокрема в управлінні ланцюгами постачання та розподілі ресурсів[1, 2]. Використання сучасних методів, таких як генетичні алгоритми та інші метаевристичні підходи, відкриває можливості для отримання кращих рішень порівняно з традиційними методами оптимізації, що особливо важливо в контексті дедалі складніших логістичних процесів.

1.2 Класифікація та методи розв'язання транспортних задач

Транспортні задачі - це один із фундаментальних класів задач дискретної оптимізації, спрямованих на визначення найкращого способу розподілу потоків між постачальниками та споживачами з мінімальними загальними витратами. Оскільки це особливий випадок задач лінійного програмування, вони мають особливу структуру, що дозволяє застосовувати спеціалізовані методи, що забезпечують високу обчислювальну ефективність. Спочатку транспортні задачі виникли в контексті транспортного планування, нині широко використовуються в логістиці, виробничих системах, телекомунікаціях, розподілених обчисленнях та інших сферах.

Класична формулювання транспортної задачі полягає у мінімізації витрат на транспортування однорідного ресурсу від набору постачальників до групи споживачів. Водночас відомі обсяги попиту і пропозиції, а також вартість транспортування одиниці ресурсу між кожним джерелом і пунктом призначення[2]. Якщо умова балансу виконана (сума пропозицій дорівнює сумі потреб), задача дозволяє прийнятне рішення і може ефективно розв'язатися за допомогою лінійного програмування, включаючи спеціалізовані алгоритми - наприклад, метод північно-західного кута, метод мінімальних елементів і метод

потенціалу.

Однак у реальних прикладних завданнях частіше зустрічаються незбалансовані формулювання, де загальна пропозиція не дорівнює сукупному попиту. Такі випадки типові для логістичних систем, де деякі ресурси можуть залишатися невикористаними або, навпаки, потреби перевищують наявні обсяги. У таких випадках завдання модифікують за рахунок додавання фіктивного постачальника або споживача, що дозволяє зберегти застосовність класичних методів, але ускладнює інтерпретацію отриманих рішень і може знизити їхню стійкість до змін у вхідних даних.

Багатоіндексні транспортні задачі мають набагато складнішу структуру, в якій рух ресурсів залежить не лише від джерел і пунктів призначення, а й від додаткових факторів, таких як час, тип добутку, спосіб транспортування та інші параметри[1]. Такі задачі формалізуються у вигляді багатовимірних масивів і потребують оптимізації у високовимірних просторах. Використання класичних методів за таких умов неможливе, що вимагає застосування універсальних або евристичних алгоритмів, здатних впоратися з багатовимірною структурою та складною системою обмежень.

Існують також інші узагальнені формулювання транспортних задач. Зокрема, у стохастичних транспортних моделях параметри задачі (наприклад, обсяги постачання, попит або витрати) описуються випадковими величинами, що відображають невизначеність реальних логістичних процесів. Розв'язання таких задач здійснюється за допомогою методів стохастичного програмування, сценарного аналізу та робасної оптимізації. Крім того, динамічні транспортні задачі враховують зміну параметрів з часом, що потребує оптимізації з урахуванням часової послідовності дій.

З теоретичної точки зору, задачі транспорту стосуються задач на потоках у графах, що дозволяє використовувати методи мережевого програмування, такі як алгоритми мінімального шляху, максимального потоку, мінімального розрізу та інші[2]. Однак, коли додаються нелінійні витрати, часові вікна, пріоритизація маршрутів та інші реалістичні обмеження, задача стає NP-складною, і точне

розв'язання зазвичай неможливе за поліноміальний час.

У сучасній науковій та інженерній практиці особлива увага приділяється використанню евристичних і метаевристичних методів, які ефективно розв'язують масштабні та погано структуровані транспортні проблеми[1]. Серед них найпопулярнішими є методи еволюційних розрахунків, включно з генетичними алгоритмами, які дозволяють отримувати приблизні високоякісні розв'язки з обмеженими обчислювальними ресурсами. Це робить їх особливо популярними у задачах із високою розмірністю, численними обмеженнями та нестабільними вхідними даними.

Таким чином, транспортні проблеми, незважаючи на очевидну простоту формулювання, охоплюють широкий спектр теоретичних і прикладних галузей. Їх ефективне розв'язання вимагає інтегрованого підходу, який поєднує методи лінійного програмування, теорії графів, ймовірнісного аналізу та обчислювального інтелекту.

1.3 Порівняння методів розв'язання транспортної задачі

Оптимізація транспортних процесів є фундаментальним напрямком у прикладній математиці та операційних дослідженнях. Вибір методу розв'язання транспортної задачі визначається не лише його розмірами, а й складністю обмежень, ступенем структурованості даних, вимогами до точності, а також дозволеним часом розрахунку. Сучасна практика показує, що для досягнення найкращих результатів раціонально поєднувати традиційні підходи та сучасні евристичні методи, адаптуючи вибір алгоритму до конкретних умов задачі[2].

Класичні методи, що використовуються для розв'язання транспортних задач, базуються на суворих математичних процедурах, які дозволяють знаходити точні розв'язки у детермінованій формулюванні. Метод північно-західного кута, метод мінімальної вартості та метод потенціалу є основними представниками цієї групи. Вони демонструють високу ефективність при малих і середніх розмірах задач, а також із суворим балансом вхідних даних і наявністю лінійної структури обмежень.

Однак ці методи суттєво втрачають продуктивність і обчислювальну стабільність у міру складності моделі, включно з появою додаткових обмежень, дисбалансу та недопустимості базових розв'язків.

Одним із ефективних інструментів для точної оптимізації транспортних задач є використання промислових лінійних програмних розв'язувачів, таких як HiGHS[4]. На відміну від класичних ручних алгоритмів, цей метод реалізує сучасні версії симплексного методу - внутрішнього точкового та розгалуженого та граничного, демонструючи високу точність, масштабованість і продуктивність. Метод HiGHS особливо актуальний при розв'язанні незбалансованих задач, де обсяги постачання та вимоги не збігаються, що типово для реальних логістичних сценаріїв і вимагає суворого врахування штрафних витрат або фіктивних вузлів.

Окрім точних методів, евристичні та метаевристичні алгоритми відіграють значну роль у розв'язанні задач транспорту, які використовуються для отримання приблизних розв'язків за умов високої розмірності або часткової невизначеності початкових даних. Генетичні алгоритми, алгоритми мурах, алгоритми рою частинок та інші стохастичні підходи активно розвиваються останніми десятиліттями і дозволяють ефективно знаходити розв'язки в умовах, де традиційні методи стають непрактичними через надмірну обчислювальну складність. Головна перевага метаевристики - здатність долати локальні мінімуми та адаптуватися до структури задачі, хоча вони не гарантують пошук глобального оптимуму[3]. Порівняльні характеристики методів наведені в таблиці 1.1

Таблиця 1.1 – Порівняння методів розв'язання транспортних задач

Метод	Переваги	Недоліки	Складність	Сфера застосування
Метод північно-західного кута (ПЗК)	Простота реалізації, швидке отримання допустимого базисного рішення	Не завжди дає оптимум, потребує подальшої оптимізації	O(m+n)	Початковий етап, базове рішення для малих завдань

Продовження табл. 1.1

Метод мінімальної вартості	Простота реалізації, більш вигідне базове рішення порівняно з ПЗК	Не гарантує оптимальність, вимагається дооптимізація	O(m·n)	Малі й середні задачі, побудова початкового рішення
Метод потенціалів	Формалізація, точність рішення	Обчислювально затратний при великих размірностях	$O(m \cdot n^2)$	Малі й середні задачі
Метод розподілу	Не потребує початкового плану, може дати глобальний оптимум	Складний в реалізації, чутливий до даним	$O(m \cdot n^2)$	Уточнення рішень, задачі середньої складності
Мурашиний алгоритм	Уникає локальних мінімумів, підходить для маршрутизації	Висока обчислювальна складність, потребує налаштування параметрів	Висока	Складні, динамічні й машрутизаційні задачі
Генетичний алгоритм	Універсальність, робота з обмеженнями, масштабуємось	Залежність від параметрів, немає гарантії оптимальності	Висока	Великі задачі, задачі з множиною обмежень
Розв'язувач HiGHS	Висока точність, промислова якість, підтримка незбалансованих задач	Менша гнучкість у нестандартних обмеженнях, чутливість до формалізації	Залежить від методу (LP/ILP)	Промислові задачі, великі й незбалансовані моделі

1.4 Еволюційні алгоритми та їх застосування в логістиці

Еволюційні алгоритми (ЕА) - це клас стохастичних методів оптимізації, заснованих на принципах природного відбору, генетичної рекомбінації та мутацій, які дозволяють ефективно розв'язувати задачі з високою розмірністю та складними обмеженнями[3]. Цей підхід є універсальним інструментом для пошуку приблизних оптимальних рішень у широкому спектрі прикладних галузей, включно з логістикою.

Основна ідея еволюційних алгоритмів полягає в моделюванні біологічних процесів еволюції, де популяція кандидатів на рішення (індивідів) поступово покращується протягом наступних поколінь. Кожна особа представлена як хромосома, структура якої кодує потенційне рішення проблеми.

Ключовими компонентами еволюційних алгоритмів є відбір, кросовер і мутація. Процес відбору забезпечує відбір рішень із найвищою пристосованістю, що сприяє посиленню впливу якісних варіантів і цілеспрямованому покращенню популяції. Рекомбінація, проведена шляхом кросовера, дозволяє поєднати генетичний матеріал двох або більше батьківських рішень, що породжує нові, потенційно більш оптимальні комбінації ознак. Впровадження випадкових змін, тобто мутацій, зберігає генетичне різноманіття і допомагає алгоритму уникати застрягання в локальних оптимумах.

Логістичні задачі, що характеризуються високою складністю, множинними обмеженнями та варіативністю вихідних даних, вимагають гнучких і адаптивних методів їх розв'язання. Еволюційні алгоритми демонструють переваги в цьому контексті завдяки своїй стійкості до локальних мінімумів[4]. Крім того, сучасні обчислювальні платформи дозволяють реалізовувати ці методи паралельно, що особливо важливо при вирішенні масштабних задач.

Практичне застосування еволюційних алгоритмів у логістиці охоплює низку напрямків, таких як оптимізація маршрутів доставки, планування транспорту, розподіл ресурсів між вузлами транспортної мережі та управління запасами. Наприклад, при розв'язанні проблеми маршрутизації транспортних засобів

еволюційні алгоритми дозволяють знаходити рішення, які мінімізують загальну вартість транспортування, одночасно задовольняючи багато логістичних обмежень. Водночас гнучкість алгоритму забезпечує адаптацію до динамічних змін транспортних і попитних умов.

Низка досліджень (Goldberg, 1989; Michalewicz, 1996) демонструють, що еволюційні алгоритми, включно з генетичними, успішно застосовувалися до задач оптимізації, пов'язаних із логістичними процесами[5]. Зокрема, їхня здатність поєднувати жадібні евристики з глобальним пошуком оптимального рішення дозволяє отримувати високі показники ефективності у порівнянні з традиційними методами, такими як метод північно-західного кута або метод потенціалу.

Також використання еволюційних алгоритмів у логістиці відображається в діяльності низки російських компаній, які прагнуть підвищити ефективність управління маршрутами, розподілу ресурсів і оптимізації витрат. Завдяки гнучкості та здатності враховувати складні обмеження, еволюційні алгоритми дозволяють адаптувати рішення до специфіки кожного логістичного процесу.

На ринку спостерігається тенденція впроваджувати хмарні оптимізаційні платформи, які реалізують цей підхід. Завдяки використанню алгоритмів, які враховують такі параметри, як дорожня ситуація, щільність трафіку та регіональні особливості, клієнти компанії досягають зниження вартості пального, скорочують час доставки та покращують якість обслуговування. Крім того, активно розробляється нішеві рішення у сфері логістики, зосереджених на оптимізації роботи складських комплексів, диспетчеризації та автоматизації останніх процесів. У цьому контексті малі та середні IT-компанії розробляють спеціалізовані системи TMS (Transport Management System)[4], які базуються на еволюційних алгоритмах, що дозволяють їм адаптуватися до змінних умов транспортного середовища в реальному часі. Такі системи демонструють високу адаптивність: оператори мутацій і перетину генерують нові комбінації рішень, що дозволяє алгоритмам успішно виходити з локальних оптимумів і знаходити ефективніші маршрути при зміні параметрів міської інфраструктури.

Використання еволюційних алгоритмів у логістиці характеризується низкою

значних переваг. По-перше, алгоритмічна адаптивність забезпечує стійкість до зовнішніх змін, що особливо важливо в умовах динамічного міського трафіку та сезонних коливань попиту. По-друге, здатність швидко отримувати приблизні оптимальні рішення може суттєво зменшити обчислювальні витрати та час, необхідний для планування маршруту, що призводить до зниження загальних операційних витрат. По-третє, хмарні рішення з відкритими API полегшують інтеграцію оптимізуючих платформ із існуючими CRM- та TMS-системами, забезпечуючи комплексне управління ланцюгом постачання[6].

Таким чином, досвід компаній, що розробляють платформи хмарної оптимізації, свідчить про високу практичну важливість еволюційних алгоритмів у логістиці. Використання цих методів дозволяє оптимізувати складні транспортні та логістичні процеси з урахуванням численних обмежень і специфіки регіонального ринку, що веде до зниження витрат, підвищення ефективності розподілу ресурсів і ефективності прийняття рішень. У майбутньому розробка та адаптація еволюційних алгоритмів сприятиме зміцненню конкурентоспроможності малого та середнього бізнесу, роблячи їх незамінним елементом сучасної системи управління логістикою.

1.5 Переваги та обмеження генетичного алгоритму

Розглянувши особливості використання генетичного алгоритму в транспортних задачах, рекомендується зосередитися на його ключових характеристиках, які впливають на якість і стабільність отриманих рішень. Цей метод, широко застосовуваний у задачах комбінаторної оптимізації, демонструє низку властивостей, які забезпечують його актуальність у контексті моделювання логістичних систем.

По-перше, генетичний алгоритм вирізняється здатністю працювати в умовах неповної або неформалізованої інформації про структуру задачі. На відміну від суворих математичних методів, він не вимагає лінійності цільової функції чи обмежень, що дозволяє ефективно використовувати його у задачах із додатковими

параметрами, нестандартними залежностями або багатоіндексною структурою. Така гнучкість дозволяє інтегрувати її у модель, описану в попередньому розділі, без суттєвої переробки алгоритму. Також важливо відзначити здатність досліджувати простір рішень за допомогою стохастичних операторів - мутації та кросовера. Це дозволяє уникнути передчасної збіжності та підвищує ймовірність знаходження приблизного глобального розв'язку, особливо у випадках, коли простір допустимих опцій містить багато локальних крайнощів. У контексті завдань реального транспорту, таких як планування доставок з урахуванням часу та вартості, ця властивість стає критично важливою[4].

Додатковим фактором, що підтверджує застосування генетичного алгоритму, є його висока масштабованість. Завдяки можливості паралельного генерування та обробки популяцій, що приймають рішення, метод залишається ефективним навіть при збільшенні розмірності вхідних даних. Ця якість особливо потрібна у задачах, де класичні методи демонструють значне збільшення часу обчислення або втрату точності.

Водночас успішне застосування генетичного алгоритму вимагає ретельного підходу до його конфігурації. Вибір параметрів суттєво впливає на результат, такий як розмір популяції, ймовірності оператора та глибина обчислень[4]. При недостатньо збалансованій конфігурації можливий передчасний виродок популяції або надмірний час, витрачений на досягнення прийнятного результату. Крім того, ефективність методу безпосередньо залежить від правильного вибору форми представлення (кодування) рішень: у транспортних задачах помилкове кодування може призвести до генерації неправильних або неінформативних маршрутів.

Отже, генетичний алгоритм є адаптивним і потужним інструментом, особливо ефективним у випадках, коли завдання виходить за межі застосування класичних методів. Однак його ефективність безпосередньо залежить від правильної архітектури реалізації та ретельного налаштування параметрів.

1.6 Генетичний алгоритм як метод розв'язання

Генетичний алгоритм - це ітеративна евристика, яка імітує процеси природного відбору та еволюції в популяціях. Його основна ідея - послідовне генерування, відбір і трансформація набору рішень, де кожне з них інтерпретується як індивід (індивід) у популяції[6]. При розв'язанні транспортних задач - як класичних, так і мультиіндексних - кожен етап алгоритму вимагає особливого підходу, оскільки безпосередньо впливає як на правильність сформованих рішень, так і на ефективність пошуку.

На початковому етапі генерується початкова популяція. На практиці цей процес може бути реалізований як повністю випадково, так і на основі евристичних чи точних методів. Випадкове формування забезпечує високу різноманітність, але у випадку багатоіндексної транспортної задачі призводить до значних труднощів: через високу розмірність і складні обмеження значна частина розв'язків є недійсною і потребує додаткової обробки. Альтернативою є гібридний підхід, у якому частина популяції формується за допомогою рішень, отриманих методами північно-західного кута, мінімального елемента, випадкової допустимої конструкції або лінійного розв'язувача (наприклад, HiGHS). Ця стратегія дозволяє досягти збалансованого початкового розподілу: з одного боку, зберігається різноманітність, а з іншого - потенційно сильні розв'язки присутні в популяції з самого початку[7]. Порівняння описаних методів наведено в таблиці 1. 2.

Таблиця 1.2 – Порівняння методів ініціалізації популяції

Метод ініціалізації	Переваги	Недоліки
Повністю випадковий	Велика різноманітність рішень	Багато неприйнятних осіб, низька збіжність
Евристичний (ПЗК, Мін.елем)	Швидке створення прийнятних рішень	Низька варіабельність
Розв'язувач (HiGHS)	Висока якість початкових рішень	Потрібен зовнішній інструмент, низька різноманітність

Продовження табл. 1.2

Гібридний підхід	Поєднання точності та різноманіття	Підвищена складність реалізації

У контексті цієї роботи надається перевага гібридній ініціалізації, що є виправданим як з точки зору обчислювальної стабільності, так і з точки зору прискореної збіжності.

Після генерації індивідів необхідно оцінити їхню пристосованість, тобто відповідність цільовій функції завдання. У класичних транспортних задачах оцінка може базуватися виключно на загальній сумі витрат, але в багатоіндексній формулі слід враховувати низку факторів: вартість, часові обмеження, допустимість усіх індексів, а часто додаткові умови, пов'язані з пріоритетами, логістичними вікнами або стабільністю маршруту. Водночас важливо не лише обчислити значення цільової функції, а й визначити штрафний захід за порушення обмежень. Якщо такі порушення ігноруються або штрафуються недостатньо суворо, алгоритм може почати концентруватися на формально дешевих, але неприйнятних рішеннях. Тому розроблений підхід використовує модифіковану функцію пристосованості, яка включає як нормалізовані значення цільових критеріїв, так і адаптивну систему штрафів. Це дозволяє забезпечити універсальність при роботі з задачами різних розмірів і структур, не перевантажуючи обчислювальний процес надмірними перевірками при кожному поколінні.

Модифіковану функцію пристосованості можна записати як[7]:

$$F(x) = C(x) + \sum_{i}^{N} \lambda_i \cdot \phi_i(x) \qquad (1.1)$$

де: $C(x)$ – значення цільової функції для рішення x, N – кількість обмежень, $\phi_i(x)$ – функція штрафу за порушення і-го обмеження, λ_i – ваговий коефіцієнт штрафу, що адаптується у процесі виконання алгоритму.

Наступний важливий крок - відбір операторів, які визначають поведінку популяції в процесі еволюції. Оператор відбору відповідає за відбір особин, які

передають свої характеристики наступному поколінню. На практиці використовуються різні підходи: рулетка (пропорційний відбір), турнірний відбір, відбір за рангом та інші. Кожен із них має як сильні, так і слабкі сторони. Наприклад, рулетка може страждати від передчасної конвергенції з сильною різницею у пристосованості, а турнірний відбір може підвищувати селективний тиск. При розв'язанні багатоіндексної задачі особливо важливий баланс між підтримкою різноманітності та посиленням відбору, оскільки надмірно агресивний відбір може швидко зруйнувати рідкісні, але потенційно цінні структури, які можуть враховувати взаємодію між кількома індексами. У рамках практичної реалізації турнірний відбір з адаптивним розміром турніру виявився найбільш стабільним.

Оператори кросоверу (рис. 1.1) відіграють ключову роль у передачі інформації між батьківськими розв'язками[8].

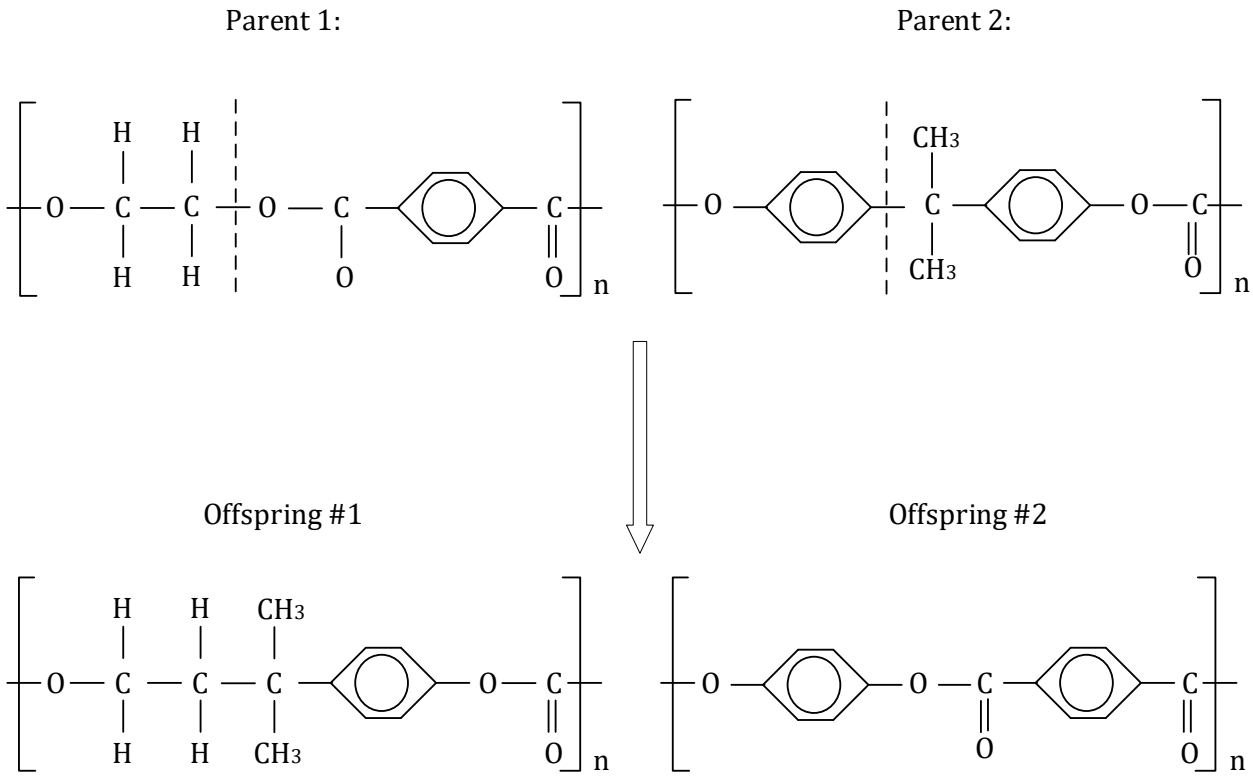


Рисунок 1.1 – Створення нащадків з батьківських рішень (стандартний одноточковий кросовер)

Для задач із числовими значеннями застосовуються як стандартні

одноточкові, так і багатоточкові кросовери, а також спеціалізовані оператори, що враховують структуру розв'язку, наприклад, методи порядку або позиційних методів. Однак для задачі з багатьма індексами основна складність полягає у збереженні валідності нащадків. Проста заміна фрагментів між батьками може призвести до порушень обмежень або дублювання розподілів, що потребує додаткових механізмів корекції. Це, у свою чергу, збільшує обчислювальне навантаження, але без цих показників кросовер втрачає ефективність. Найбільш стабільним був модифікований одноточковий кросовер із подальшою корекцією нащадків за принципом мінімального порушення обмежень[8].

Оператор мутації використовується для підтримки генетичного різноманіття та дослідження нових областей простору прийняття рішень. Випадкові зміни значень або перестановки елементів є найпоширенішими. Однак у контексті багатоіндексорної задачі важливо, щоб мутація не знищила валідність рішення. Це означає, що оператор повинен бути досить «обережним» у модифікації розв'язку в межах дозволених відхилень і обов'язково дотримуватися заданих балансів для всіх індексів. Крім того, мутація може використовувати інформацію про поточну структуру розчину, наприклад, для введення елементів локального покращення, тим самим посилюючи напрямок еволюції. У ході порівнянь на практиці найкращі результати були отримані за допомогою спрямованої мутації з обмеженням на величину змін і перевірки валідації.

Процес еволюції контролюється за допомогою стратегії умов заміни поколінь і зупинки. Заміна може бути повною (усі батьків замінюються нащадками) або частковою (деякі з найкращих рішень зберігаються). У багатоіндексній задачі збереження елітних особин особливо важливе, оскільки це можуть бути рідкісні, важко досяжні комбінації індексів[9]. Щодо умов зупинки, традиційно використовуються фіксована кількість поколінь, відсутність покращення або досягнення порогової пристосованості. На практиці, однак, комбінований підхід корисний у багатоіндексній задачі, дозволяючи алгоритму завершити як стагнацію, так і коли вичерпається обчислювальний бюджет.

Разом ці кроки становлять основу адаптивного генетичного алгоритму,

здатного впоратися з високою розмірністю та структурною складністю багатоіндексних транспортних задач. Вибір конкретних операторів і параметрів вимагає емпіричного налаштування та повторного тестування, оскільки універсальних рішень немає. Однак вже на рівні теоретичного аналізу очевидно, що правильна реалізація кожного етапу алгоритму є критичною для отримання робочого та ефективного інструменту.

Висновки до розділу:

У першому розділі проведено комплексний аналіз логістичних задач та методів їх розв’язання. Розглянуто особливості логістичних систем, зокрема транспортних задач, які характеризуються необхідністю оптимального розподілу ресурсів за наявності обмежень і значної варіативності вхідних даних. Встановлено, що такі задачі мають високу практичну значущість і широко застосовуються в реальних умовах функціонування підприємств.

Проаналізовано класифікацію транспортних задач та основні підходи до їх розв’язання. Здійснено порівняння класичних методів оптимізації, що дозволило виявити їх ефективність для задач невеликої розмірності, а також обмеження при застосуванні до складних, багатоіндексних і незбалансованих задач. Визначено, що традиційні методи часто є обчислювально затратними та недостатньо гнучкими для врахування особливостей реальних логістичних процесів.

Окрему увагу приділено еволюційним алгоритмам, зокрема генетичному алгоритму, як перспективному підходу до розв’язання задач оптимізації. Розглянуто принципи його роботи, переваги та обмеження. Встановлено, що генетичний алгоритм забезпечує високу адаптивність, можливість роботи зі складними просторами рішень та ефективність при розв’язанні задач великої розмірності.

Таким чином, теоретичний аналіз підтвердив актуальність задач оптимізації транспортних процесів у логістиці, виявив обмеження класичних методів і обґрунтував доцільність використання генетичного алгоритму як ефективного інструменту їх розв’язання. Отримані результати визначають напрям подальших

досліджень, пов’язаних із розробкою та реалізацією алгоритмічного забезпечення для розв’язання транспортних задач.

У наступному розділі буде розглянуто структуру розроблюваної системи, прийняті проєктні рішення, а також проведено аналіз ефективності генетичного алгоритму у порівнянні з класичними та сучасними підходами, включаючи промислові розв’язувачі.

РОЗДІЛ 2
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО РІШЕННЯ

2.1 Обґрунтування обраного програмного забезпечення

Використання мови програмування Python: Вибір мови програмування Python обумовлений її широким застосуванням у задачах наукових обчислень, оптимізації та аналізу даних[11].

Наявність потужних бібліотек:

– NumPy - для роботи з багатовимірними масивами (критично для ND транспортної задачі)

– SciPy - реалізація методів лінійного програмування (зокрема HiGHS)

– Pandas - обробка експериментальних даних

Зручність прототипування: Дозволяє швидко реалізовувати та модифікувати алгоритми (генетичні, евристичні, точні).

Підтримка багатовимірних структур: Це критично, оскільки розглядається не тільки класична 2D транспортна задача, а й її узагальнення на N-вимірний випадок.

Використання Tkinter: Бібліотека Tkinter використовується для створення графічного інтерфейсу користувача (GUI) та нее потребує додаткових залежностей, що спрощує розгортання системи.

Підтримка інтерактивної роботи:

- завантаження задач (JSON);
- запуск алгоритмів;
- візуалізація результатів.

Використання у дослідницьких системах, GUI дозволяє:

- проводити експерименти без зміни коду;
- демонструвати результати.

Мінімальні накладні витрати у порівнянні з більш важкими фреймворками (Qt, web UI)

Використання Matplotlib. Бібліотека Matplotlib застосовується для побудови графіків.

Стандарт де-факто для наукової візуалізації в Python

Можливість побудови всіх необхідних графіків:

- залежність вартості від розмірності;
- збіжність генетичного алгоритму;
- порівняння методів;
- boxplot (стабільність);

Гнучкість оформлення: дозволяє адаптувати графіки.

Інтеграція з GUI: Через FigureCanvasTkAgg - вбудування графіків прямо в інтерфейс.

Використання Microsoft Excel. Збереження результатів експериментів у форматі Microsoft Excel обумовлене наступним:

- Стандарт для представлення табличних даних;
- Широко використовується у наукових та прикладних дослідженнях;
- Зручність аналізу:

  сортування

  фільтрація

  побудова додаткових графіків

Сумісність з іншими системами: дані можуть бути легко імпортовані в статистичні пакети або системи обробки даних

Підтримка великих обсягів експериментів: результати batch-тестування.

Обраний стек технологій буде виглядати наступним чином:

- Python → алгоритми та обчислення;
- Tkinter → інтерфейс користувача;
- Matplotlib → наукова візуалізація;
- Excel → збереження та аналіз результатів

утворює цілісну дослідницьку платформу, яка забезпечує: генерацію задач та їх автоматичне розв’язання; проведення серії експериментів; аналіз та візуалізацію

результатів. Діаграма використання (Use Case) представлено на рис. 2.1

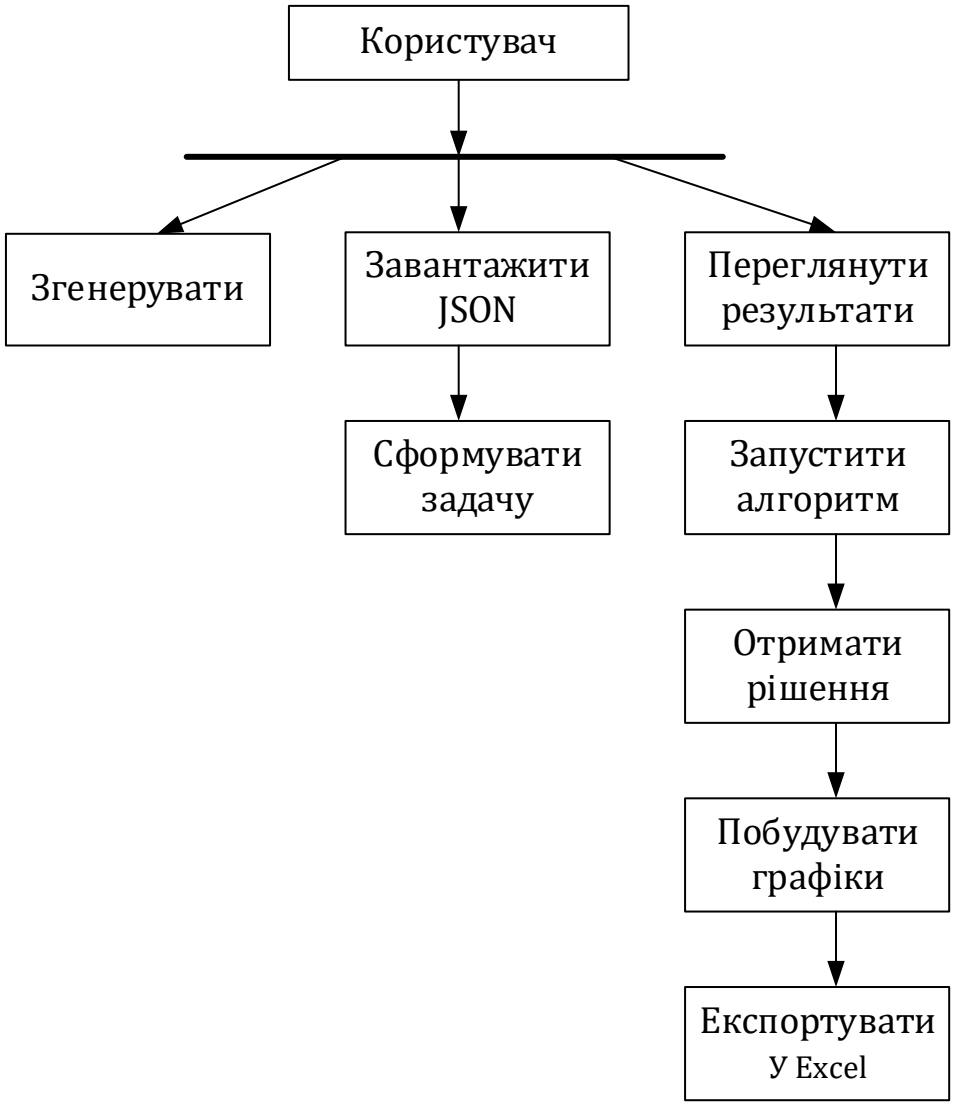


Рисунок 2.1 - Діаграма використання для користувача

Використання зазначених технологій забезпечує баланс між обчислювальною ефективністю, гнучкістю розробки, наочністю представлення результатів та відтворюваністю експериментів, що є критично важливим для проведення наукових досліджень у галузі оптимізації транспортних задач.

2.2. Опис ключових компонентів програмного коду

Програмна реалізація побудована на модульному принципі, що забезпечує читабельність, масштабованість і повторне використання компонентів. Основний

код реалізований на Python і включає як графічний інтерфейс користувача, так і обчислювальний рушій, який містить методи для створення рішень, перевірки обмежень, виконання генетичного алгоритму та аналізу результатів.

Структура проєкту представлена у вигляді окремих логічно розділених модулів. Головний файл запускає інтерфейс і пов'язує дії користувача з відповідними функціями. Основні розрахунки поділені на окремі модулі: один відповідає за обробку вхідних даних і побудову структури задачі, інший - за реалізацію алгоритмів, а третій - за експорт даних і побудову графіків. Це гарантує, що логіка візуалізації ізольована від обчислювальної логіки.

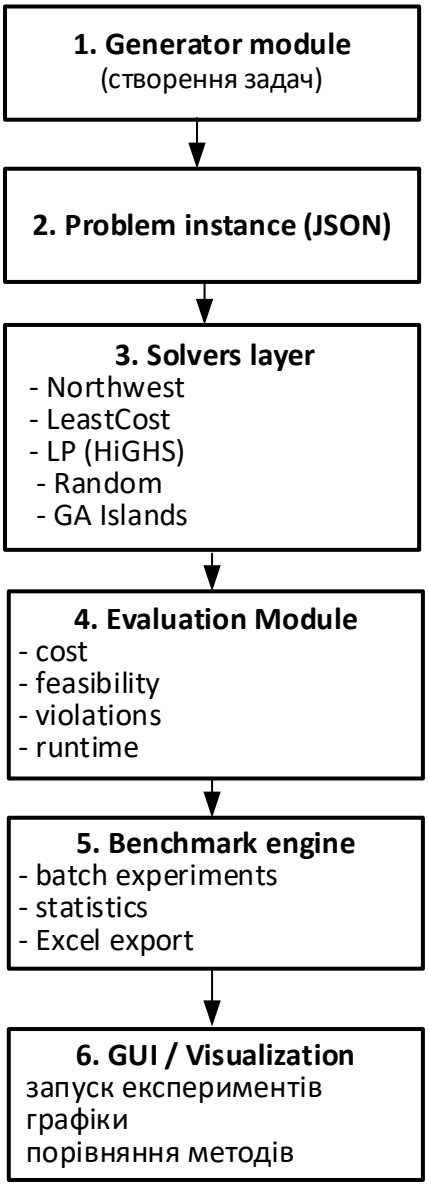


Рисунок 2.2 - Архітектура системи

Модуль обробки даних містить функції створення багатовимірних тензорів, які описують структуру задачі. Незалежно від кількості вимірів, вхідні дані конвертуються в єдиний формат, який підтримує валідацію обмежень і створення дійсних маршрутів. Було впроваджено систему масок для виключення неприйнятних напрямків.

Класичні методи розв'язання, такі як метод північно-західного кута, мінімального елементу, випадкової генерації дозволеного плану й розв'язувача HiGHS, реалізовані як окремі функції з одним інтерфейсом. Це дозволяє запускати будь-яку з них як окремо, так і як частину генерації початкової популяції генетичного алгоритму. HiGHS викликається через формат моделі LP, який автоматично генерується з поточної структури завдань[10].

Центральною частиною обчислень є модуль генетичного алгоритму. Він реалізує функції генерації популяції, оцінки пристосованості, відбору, кросовера, мутації та відбору елітних особин. Хромосоми представлені як одномірні масиви з подальшим перетворенням у багатовимірну структуру при перевірці обмежень і розрахунку вартості. Використовується адаптивна система штрафів, а також модифіковані оператори мутацій і перетину, які забезпечують допустимість нащадків навіть у багатоіндексних задачах. Усі параметри алгоритму (розмір популяції, ймовірності, стратегія елітарності, кількість поколінь) доступні для зміни через інтерфейс[13].

Впроваджено острівну модель, яка дозволяє поділити населення на кілька незалежних підгруп. Кожна група розвивається окремо, і після певного інтервалу поколінь обмінюються найкращими рішеннями. Механізм реалізується на рівні ітеративної координації локальних популяцій із контролем конвергенції.

Модуль візуалізації та експорту генерує підсумкові таблиці, побудовує графіки змін у цільовій функції та порівняння методів, а також генерує файли Excel з результатами. Для графів використовується бібліотека matplotlib, а для експорту таблиць - openpyxl. Програма автоматично генерує резюме з мінімальними значеннями вартості кожного методу і дозволяє порівнювати їх у табличному та графічному вигляді[12].

Графічний інтерфейс реалізований на основі tkinter. вікна автоматично адаптуються до розмірності завдання: відповідні поля для багатовимірних індексів додаються або приховуються. Існують функції для завантаження та збереження даних, покрокового виконання алгоритму, відображення поточного транспортного плану, зупинки обчислення та виправлення результату. Користувач може встановлювати параметри алгоритму, змінювати методи розв'язання, отримувати миттєву візуалізацію прогресу та експортувати звіти.

Програмний код структурований так, що його легко підтримувати, змінювати та масштабувати. Усі функції мають пояснювальні коментарі. Архітектура орієнтована на розширення: можна додавати нові методи, обмеження, стратегії вибору та візуалізацію без необхідності переписувати логіку системи.

2.3 Архітектура програмного забезпечення та особливості її реалізації

Розробка програмного рішення для транспортної проблеми, включаючи її багатоіндексні модифікації, вимагала створення гнучкої та масштабованої архітектури, здатної адаптуватися до різних розмірів даних, методів розв'язання та критеріїв оцінки. Ключовою метою цієї реалізації було поєднання класичних алгоритмів і сучасних еволюційних підходів в одному програмному продукті, з можливістю порівняння їхньої якості та адаптації до практичних сценаріїв. Особливу увагу приділялося підтримці довільної розмірності завдання, а також зручності взаємодії користувача через графічний інтерфейс.

У ході реалізації було створено універсальну структуру, яка дозволяє вирішувати як класичні двовимірні, так і багатоіндексні транспортні задачі. Узагальнюючи модель, можна було звести обробку даних до однієї форми, що забезпечувало послідовну логіку для всіх методів розв'язання, що використовувалися в системі. Це не лише спростило підтримку коду, а й забезпечило стабільний результат, включно з експортом у таблиці, візуалізацією транспортних планів і побудовою порівняльних графіків.

Далі ми детально розглянемо структуру представлення даних, особливості

кодування рішень для генетичного алгоритму, методи генерації початкових рішень, реалізацію операторів і механізмів еволюції, а також взаємодію компонентів через користувацький інтерфейс. Сам код програми наведено в Додатку А і буде розглянутий у цьому розділі лише загалом.

Подача початкових даних у програмі базується на єдиному універсальному форматі, здатному описувати як класичні, так і узагальнені транспортні завдання. Вхідні дані включають: розмір задачі, структуру індексів, масив витрат на перевантаження, обсяги постачання та споживання, а також додаткові параметри, що визначають обмеження або критерії оптимізації[11]. Для підтримки багатоіндексних завдань використовується багатовимірна структура зберігання у вигляді тензора довільної розмірності, де кожна вісь відповідає одному з вимірів - наприклад, відправнику, отримувачу, проміжному складу, типу вантажу або часовій позначці.

У класичному випадку двовимірна задача кодується як матриця: кожна клітинка містить вартість транспортування від одного джерела до одного споживача. У узагальненій формулюванні розв'язок подається у вигляді тензора, що містить об'єм запасів для всіх можливих комбінацій індексів. Наприклад, у задачі розмірності (3×3×3×3) тензор має чотири виміри, і в кожній точці він має кількість ресурсу, що передається відповідним маршрутом.

Цей підхід вимагав розробки механізму валідації прийняття рішень, який перевіряє виконання обмежень у кожному вимірі. Для кожної осі встановлюються умови балансу (наприклад, кількість на і-індексі має відповідати постачанню з цього складу), і реалізація перевіряє їх або при генерації рішення, або після виконання операторів кросовера та мутацій. Крім того, система масок використовується для виключення недійсних маршрутів і прискорення перевірок валідації.

Хромосома в генетичному алгоритмі реалізується у вигляді одномірного масиву, який відображається у відповідний тензор за відомою розмірністю. Такий підхід дозволяє легко реалізувати оператори GA та підтримувати довільну кількість вимірювань без необхідності переписувати окремі алгоритми для

кожного випадку[11].

Програма реалізує кілька методів генерації початкових допустимих рішень, що дозволяє створювати початкові транспортні плани для класичного аналізу, а також використовувати їх як елементи початкової популяції при запуску генетичного алгоритму. Це суттєво впливає на швидкість збіжності та обчислювальну стабільність, особливо у високовимірних задачах.

Реалізація методу північно-західного кута (NWS) базується на покроковому заповненні транспортного плану, починаючи з верхнього лівого кута багатовимірної таблиці. Алгоритм послідовно розподіляє обсяг постачання та споживання, рухаючись уздовж осьових координат, доки відповідні ресурси не вичерпаються. У багатоіндексному завданні реалізація SCA стає складнішою: потрібно пріоритезувати порядок проходження індексів, щоб забезпечити передбачувану та коректну поведінку на будь-якій кількості вимірів[14]. У програмі це реалізовано як ітерація координат у лексикографічному порядку, що зберігає алгоритмічну простоту та розширює застосовність.

Метод найменшої вартості базується на послідовному відборі комірок із найнижчою вартістю серед тих, що ще не розподілені, що, як правило, забезпечує більш прибутковий стартовий план з точки зору цільової функції. Для багатовимірного випадку було реалізовано механізм для пріоритетного сортування всіх координатних позицій за вартістю, а потім послідовного розподілу ресурсів. Цей метод, як і NWS, гарантує допустимість рішень і вводить початкове різноманіття в популяцію GA.

Крім того, реалізовано генератор випадкових дійсних розв'язків. На відміну від наївного випадкового генерування, яке призводить до багатьох неприйнятних рішень, цей генератор використовує обмежений перерозподіл ресурсу між дійсними маршрутами з суворим контролем балансу над усіма вимірами. Це дозволяє формувати прийнятних, але різноманітних особин, придатних для включення в популяцію без необхідності подальшої фільтрації.

Система також включає інтерфейс для взаємодії з зовнішнім LP HiGHS-розв'язувачем, який забезпечує побудову еталонного транспортного плану за

допомогою лінійного програмування[13]. Це дозволяє не лише порівнювати точні та приблизні методи, а й використовувати отримане рішення як одне з найякісніших джерел для початкової популяції в GA.

Поєднання класичних, евристичних і точних методів генерації рішень забезпечує гнучкість на початковому етапі моделювання та закладає основу для більш стабільного та швидшого еволюційного компонента.

Центральною частиною програмної системи є реалізація генетичного алгоритму, адаптованого для роботи з транспортними задачами різних розмірів. Алгоритм розроблений для забезпечення масштабованості, толерантності до локальних мінімумів і можливості тонкого налаштування параметрів для різних класів завдань.

Кожен розв'язок у популяції кодується як одномірний масив фіксованої довжини, що відповідає кількості комірок у багатовимірній структурі задачі. Цей розв'язок потім перетворюється на тензор із заданою розмірністю за потреби для обчислення цільової функції та перевірки обмежень. Цей підхід дозволив уніфікувати обробку хромосом і спростити реалізацію операторів кросоверу та мутацій, незалежно від кількості індексів. Усі обмеження (наприклад, пропозиція, пропозиція, прийнятність маршруту та вигадані призначення) перевіряються за маскою допустимості та матрицями обмежень, створеними на етапі завантаження завдання.

Оцінка фізичної придатності реалізується як функція, що включає значення цільової функції та штрафи за відхилення від допустимості. Під час роботи алгоритму коефіцієнти штрафу можуть змінюватися: якщо кількість дійсних розв'язків занадто мала або популяція вироджується, штрафи послаблюються для розширення простору пошуку. Така адаптивність підвищує стабільність алгоритму у складних випадках.

Відбір здійснюється на основі турнірного відбору: випадковим чином відбирається кілька осіб, з яких індивід із найкращою фізичною підготовкою передається наступному поколінню. Розмір турніру, а також частка елітних учасників регулюється параметрами алгоритму. Це дозволяє гнучко балансувати

між збереженням найкращих рішень і дослідженням нових просторів.

Оператор перетину реалізується у вигляді модифікованого одноточкового кросовера. Для кожної пари батьків вибирається випадкова точка зрізу, після чого обмінюються сегменти. Оскільки така операція може призвести до порушення обмежень, після кросовера застосовується механізм корекції, який вирівнює потік уздовж осей і виключає дублювання. Крім того, реалізується перевірка валідації маршрутів з автоматичним обнуленням заборонених комірок. У більшості випадків використовується локальна напрямна мутація, тобто невеликий перерозподіл об'ємів між двома дозволеними напрямками при збереженні загального балансу. За потреби можна застосувати глобальну мутацію, тобто відтворення цілого блоку розчину, якщо популяція стагнується.

Алгоритм використовує часткову заміну поколінь: зберігається певна кількість найкращих індивідів (еліта), решта замінюється новими нащадками. Таке рішення дозволяє уникнути втрати хороших рішень і допомагає стабілізувати прогрес. Умови вимкнення включають обмеження генерації, досягнення певного рівня цільової функції та відсутність покращень у певній кількості ітерацій.

Реалізація підтримує острівну модель: популяція поділена на ізольовані підгрупи, кожна з яких еволюціонує незалежно протягом кількох поколінь. У певні моменти між островами обмінюються найкращими рішеннями. Це підвищує стійкість до локальних спадів і дозволяє використовувати різні стратегії одночасно за один захід. Кожен острів може використовувати власний метод генерації початкових рішень, включаючи результати класичних алгоритмів і LP-розв'язувач, що допомагає зберегти різноманітність і прискорити досягнення оптимальних результатів.

Для підвищення зручності роботи з системою реалізовано графічний інтерфейс користувача, який дозволяє встановлювати параметри задачі, обирати методи розв'язання, відстежувати прогрес алгоритму та візуалізувати отримані результати. Інтерфейс побудований так, щоб бути інтуїтивно зрозумілим і водночас надавати доступ до всіх функцій, необхідних для налаштування та

аналізу завдань різної складності.

На екрані введення користувач вказує розмір задачі, вводить масиви матеріалів, вимог і матрицю (або тензор) транспортних витрат. У випадку задачі з багатьма індексами поля динамічно адаптуються до кількості вимірів. Крім того, можна встановити обмеження та вигадані вказівки, а також вибрати цільову функцію (наприклад, щоб мінімізувати витрати або час).

Для початку розрахунку користувач може обрати один із доступних методів: класичний (північно-західний кут, мінімальний елемент), лінійний (HiGHS), випадковий або генетичний алгоритм. У випадку відбору GA активуються поля для встановлення таких параметрів: розмір популяції, ймовірність мутації, кількість поколінь, острівна модель, механізм відбору тощо.

Під час роботи алгоритму на екрані відображається поточний транспортний план, значення цільової функції, найкращі знайдені рішення та графік змін цільової функції за поколіннями. Функція ранньої зупинки та виправлення поточного результату також доступна. Для задач з великими розмірами доступна агрегована візуалізація, яка дозволяє швидко оцінити структуру рішення за допомогою проєкцій.

Після завершення розрахунку програма пропонує експортувати отримані дані у файл Excel, включно з фінальним транспортним планом, значенням цільової функції та порівняльним звітом про всі використані методи. Файл містить підсумкову таблицю з мінімальними витратами на кожен метод, а також графіки для порівняння ефективності різних тестових завдань. Для зручності аналізу користувач може обрати формат презентації (плоский, згорнутий, агрегований).

Завдяки такій організації інтерфейсу програму можна використовувати не лише для досліджень, а й як інструмент для підтримки прийняття рішень у логістиці. Здатність обробляти як класичні, так і багатоіндексні задачі, інтеграція з точним розв'язувачем і підтримка гібридних методів роблять розроблену систему універсальним інструментом для аналізу та оптимізації транспортних схем різної складності.

2.4 Опис експериментального налаштування та даних тестування

Опишемо методологію проведення експериментів, спрямованих на оцінку ефективності розробленого програмного продукту, що реалізує генетичний алгоритм для розв'язання транспортної проблеми. Експерименти проводилися на обчислювальному обладнанні, що відповідає типовим офісним стандартам, що дозволяє оцінити практичну застосовність алгоритму в реальних умовах.

Тестування проводилося у трьох режимах: у першому режимі тестувався інтерфейс, у другому - для збалансованої транспортної задачі, де витрати та можливості сукупно рівні, і в третьому режимі - для незбалансованої транспортної задачі, де витрати та можливості сукупно нерівні. Водночас параметри алгоритму, такі як розмір популяції та кількість поколінь, встановлюються через графічний інтерфейс, що забезпечує відтворюваність експериментів і можливість гнучко коригувати метод залежно від умов конкретного завдання.

Початкові дані для експериментів - це набори, які відображають різноманітні логістичні сценарії. Перш за все, це дані про запаси постачальників, споживчий попит і матрицю транспортних витрат. Для задачі з багатьма індексами додатково враховуються параметри, що характеризують ємність сховищ або інші обмеження, що впливають на розподіл ресурсів. Такий підхід дозволяє моделювати як прості, так і складні логістичні процеси, які відповідають реаліям сучасного бізнесу.

Результати розробленого алгоритму порівнюються за кількома ключовими показниками, включно з загальними транспортними витратами, часом розрахунку та ступенем оптимізації розподілу ресурсів. Класичні методи «мінімальної вартості» та «північно-західного кута» були обрані для порівняння, оскільки вони широко використовуються для отримання базового допустимого розв'язання транспортних задач[16]. Планується детальний аналіз для оцінки переваг генетичного алгоритму в умовах високої розмірності та складних обмежень, а також для виявлення тих сценаріїв, у яких класичний метод може

здаватися більш ефективним з точки зору робочого часу.

Очікується, що запропонований алгоритм продемонструє нижчі загальні транспортні витрати та високу адаптивність до змінних умов завдання, незважаючи на збільшення часу обчислення. Проведення експериментів не лише підтвердить практичне значення розробленого програмного продукту, а й визначить оптимальні параметри, які, у свою чергу, стануть основою для подальших удосконалень алгоритму.

2.5 Інтерфейс користувача та функції програми

Розроблений програмний продукт оснащений графічним інтерфейсом, який дозволяє виконувати оптимізацію транспортного завдання без необхідності звертатися до коду[17]. Інтерфейс універсальний: він однаково підтримує як класичне (двовимірне), так і багатоіндексне формулювання задачі, забезпечуючи рівномірну роботу незалежно від розмірності вхідних даних. Програма автоматично визначає структуру завдання та застосовує відповідні алгоритмічні процедури, що усуває потребу користувача ручно перемикати режими (див. Рисунок 2.3).

Робота з системою починається із завантаження вхідного файлу у форматі JSON, що містить матрицю вартості та масиви обмежень на кожній з осей - рисунок 2.4.

Після завантаження дані візуально відображаються у текстовому полі - межі та багатовимірна матриця вартості відображаються окремо (див. рисунок 2.5). Це дозволяє одразу переконатися, що структура завдання правильна, і, за потреби, внести зміни на етапі підготовки даних.

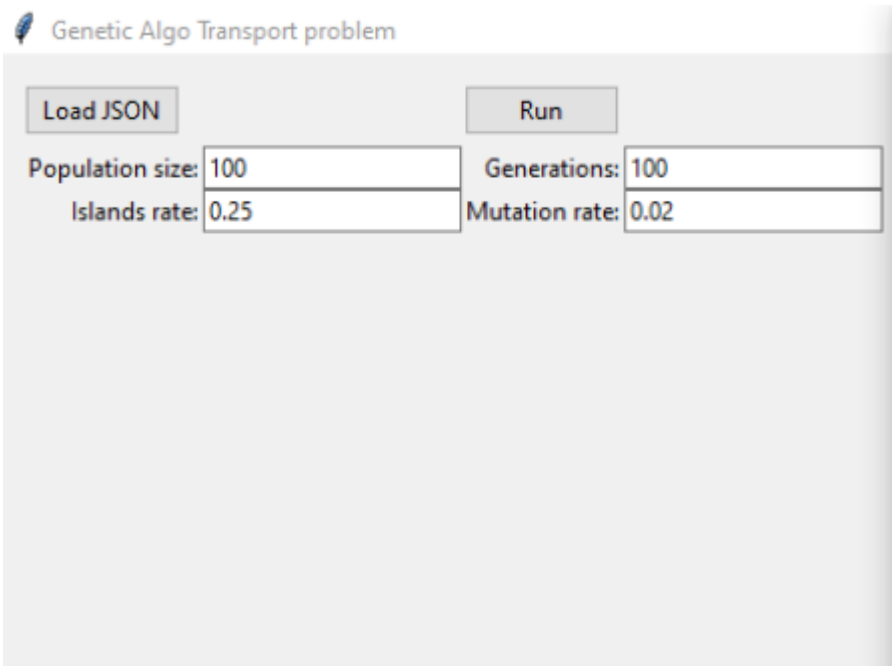


Рисунок 2.3 – Інтерфейс програми,

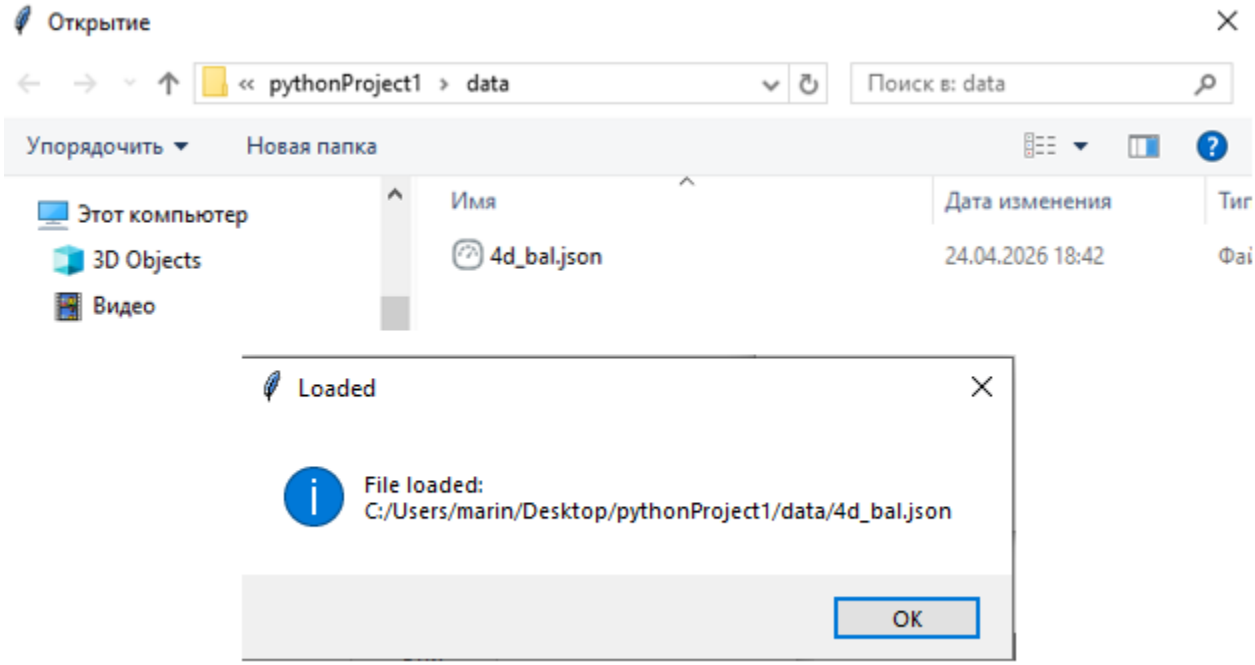


Рисунок 2.4 – Завантаження JSON-файлів

Користувач встановлює параметри генетичного алгоритму через відповідні поля (Рисунок 2.3 – 2.4: розмір популяції, кількість поколінь, ймовірність мутації та частка індивідів, створених класичними методами (так званні seed-методи).

Ці параметри визначають стратегію еволюції і дозволяють експериментувати з різними налаштуваннями, щоб побачити, як вони впливають на результат.

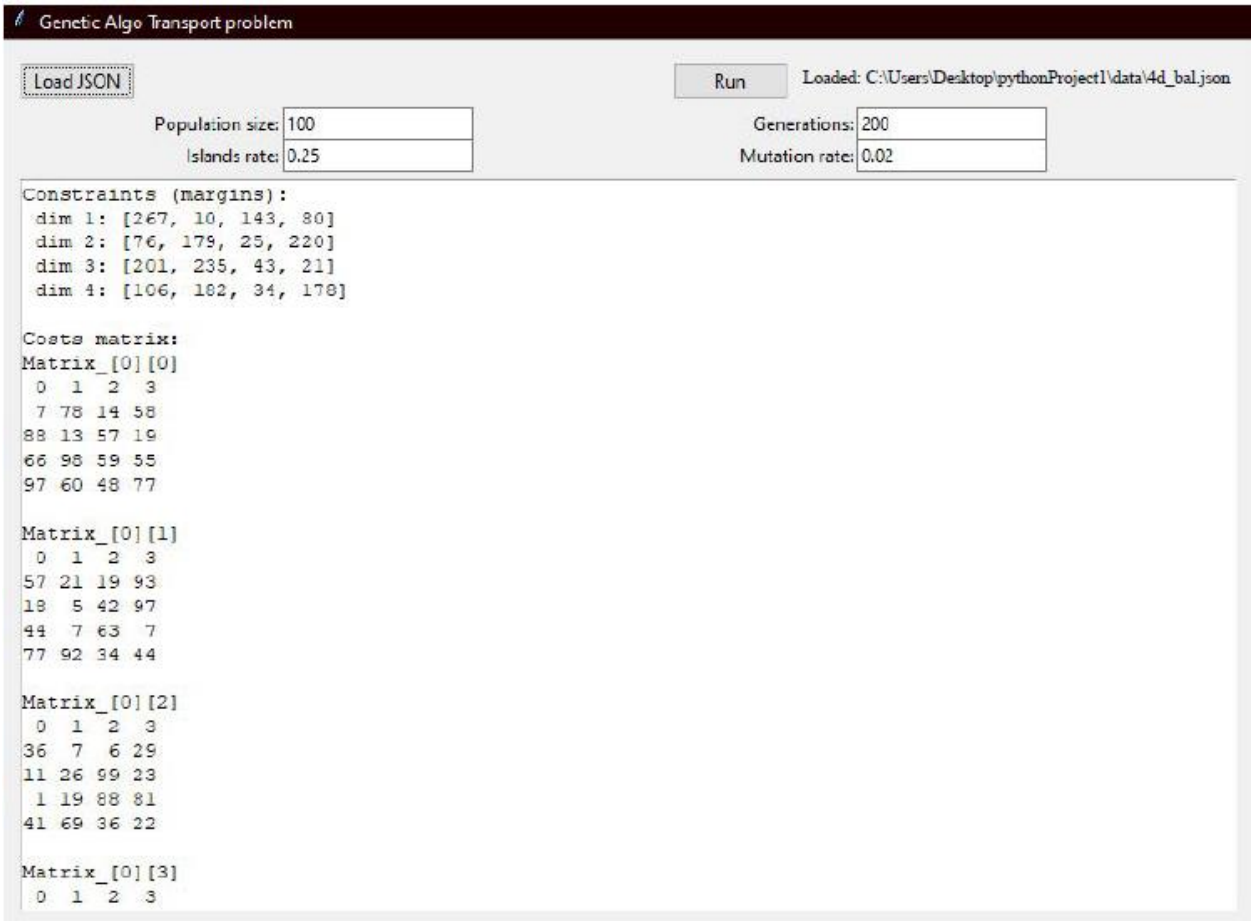


Рисунок 2.4 – Відображення завантажених даних

Алгоритм реалізований за острівною моделлю з чотирма незалежними популяціями, кожна з яких ініціалізована різним методом: північно-західний кут, мінімальний елемент, розв'язувач HiGHS або випадкове генерування – рисунок 2.5

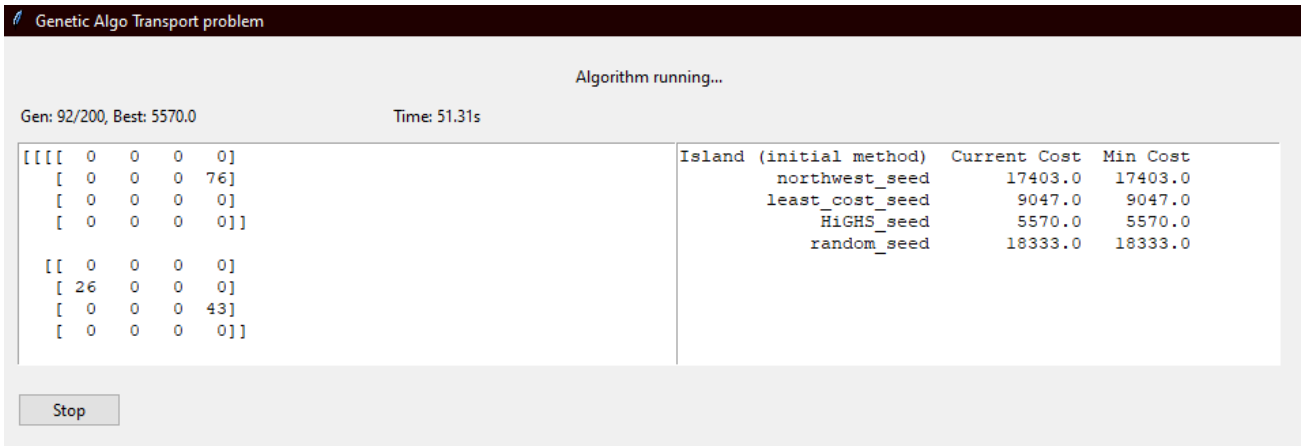


Рисунок 2.5 – Робота алгоритму

Після запуску алгоритму користувач отримує інформацію про процес еволюції: поточне число покоління, найкраще значення вартості та загальний час роботи. Водночас відображаються актуальний найкращий транспортний план і таблиця з порівнянням вартості рішень на кожному з островів. Це дозволяє відстежувати зближення островів.

Після завершення оптимізації програма генерує фінальне вікно з розширеною аналітикою (рисунок 2.6). Відображаються загальна вартість, фінальний транспортний план, перевірка відповідності обмеженням для кожної осі (включно з недонавантаженням і перевантаженням), а також порівняння з результатами, отриманими при початковій генерації класичними методами. Ця інтеграція вбудованого аналізу дозволяє безпосередньо порівнювати GA з класичними підходами без необхідності запускати їх окремо.

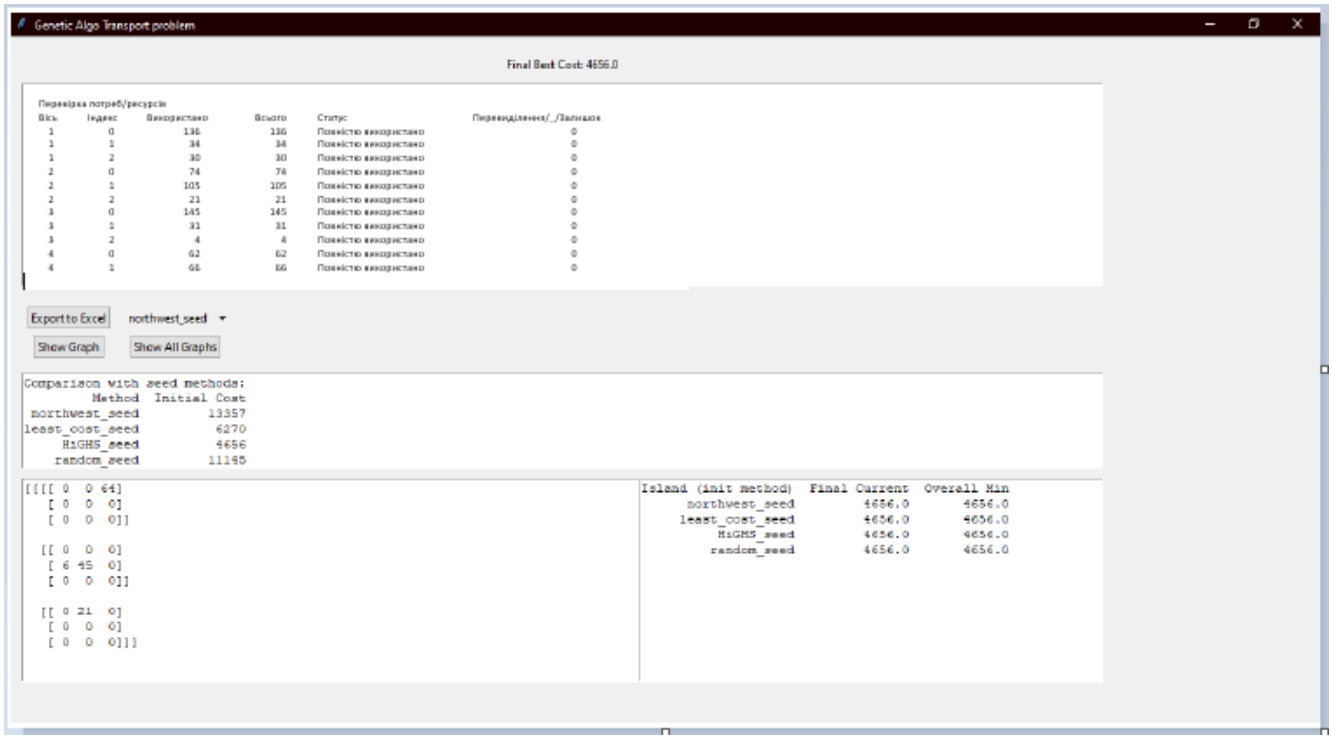


Рисунок 2.6 – Вікно з вихідними даними та аналітикою

Окрім текстової інформації (Рисунок 2.7), користувач має доступ до графіків збіжності для кожного метода окремо (Рисунок 2.8) або для всіх одночасно (Рисунок 2.9).

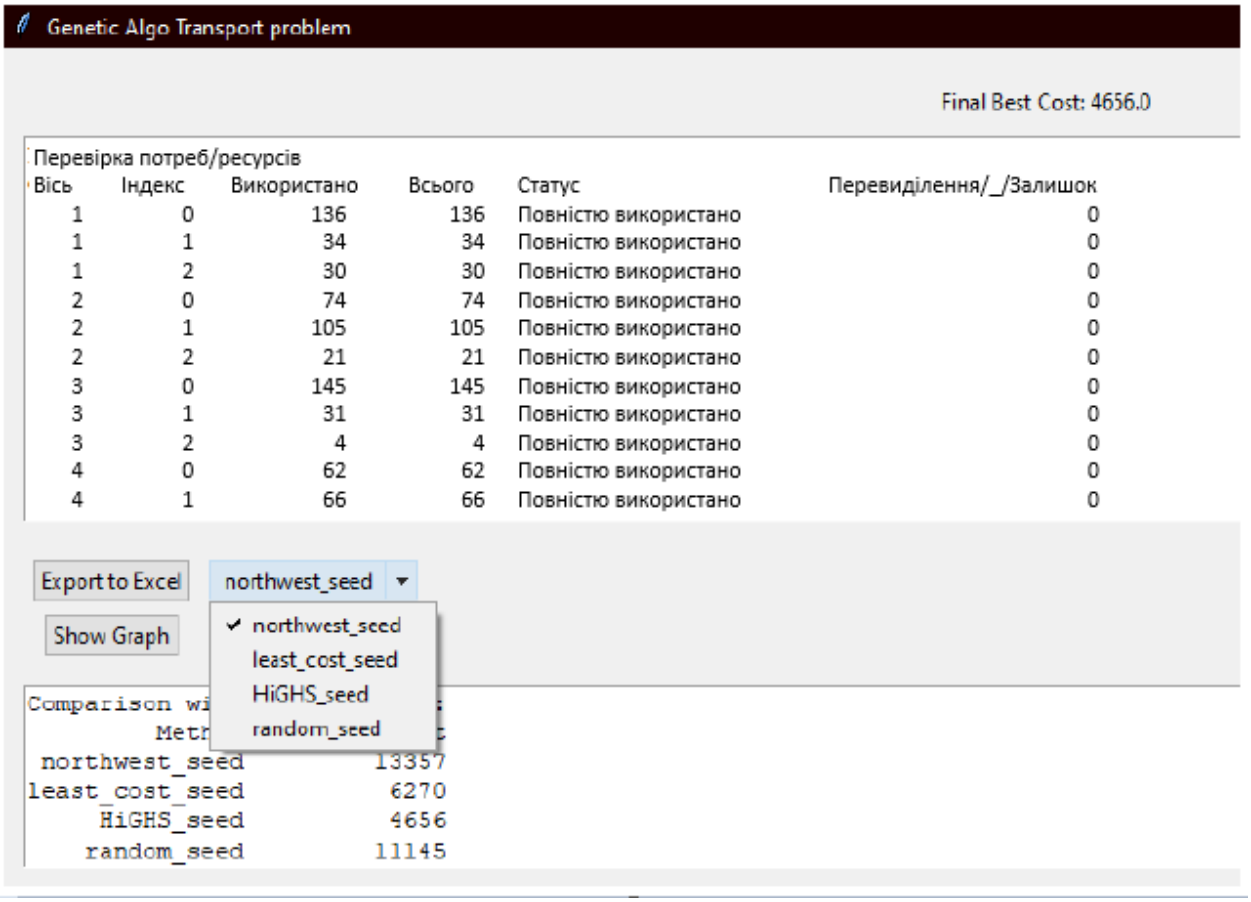


Рисунок 2.7 – Вибір графіка для відображення

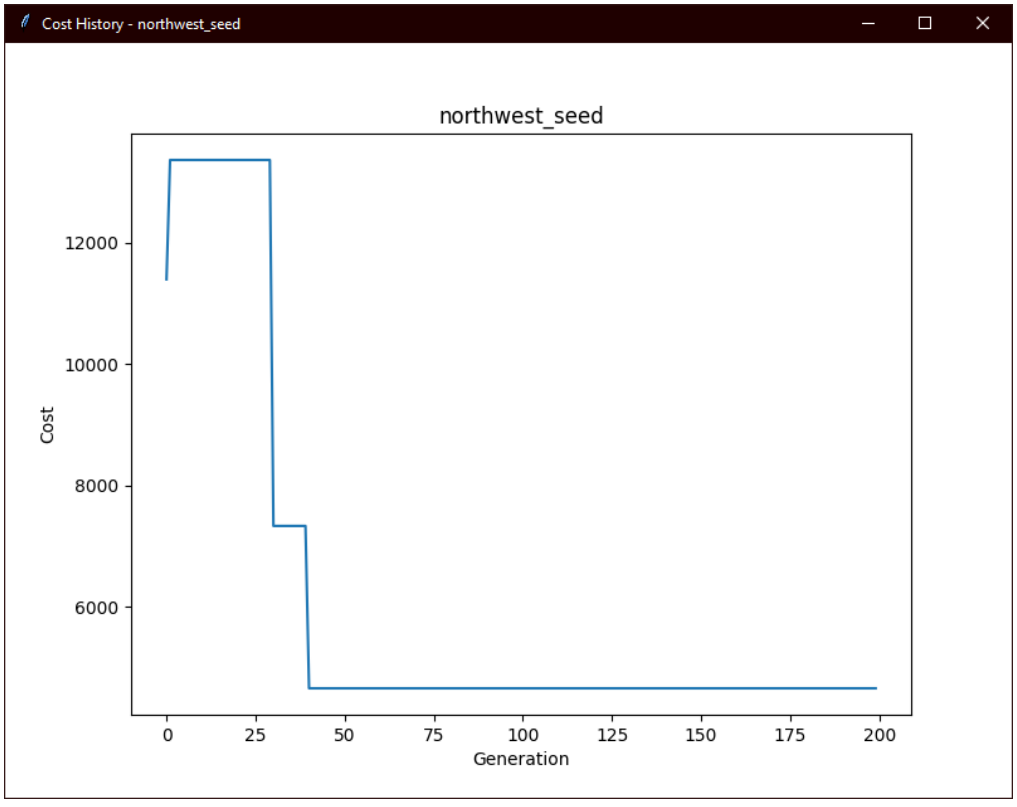


Рисунок 2.8 – Графіки мінімальних витрат методу ПЗК

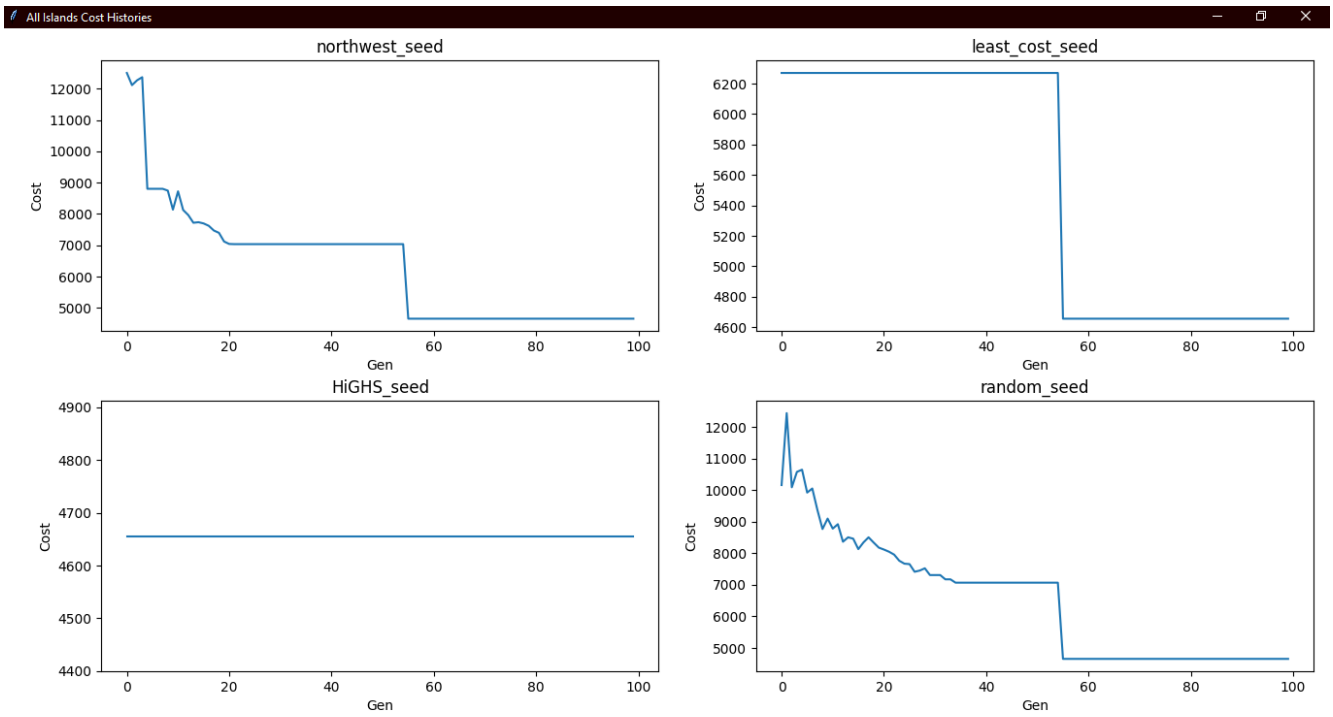
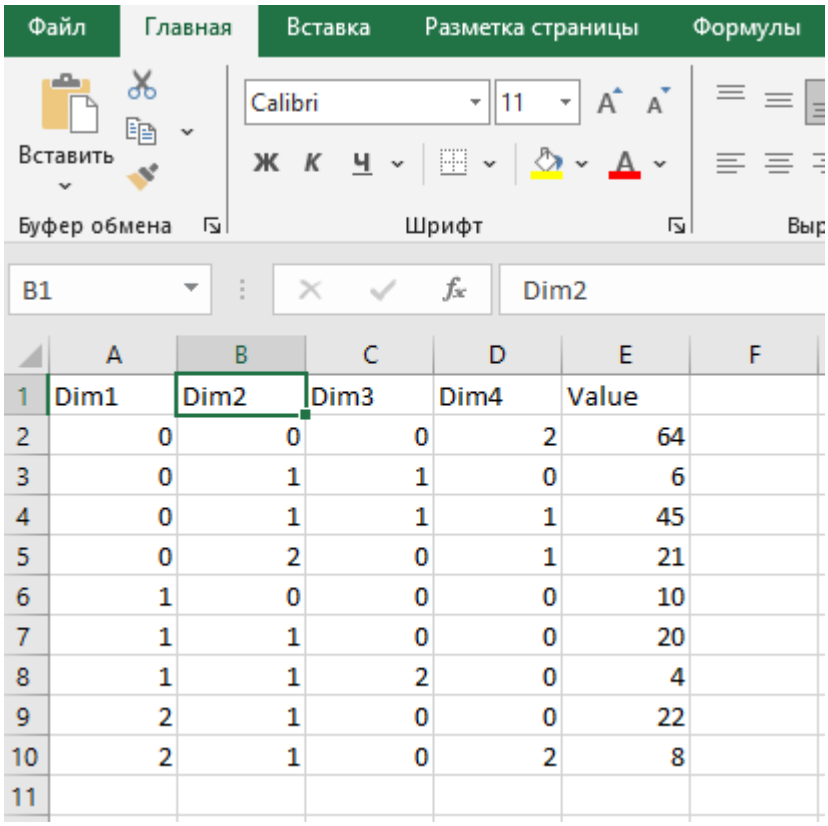


Рисунок 2.9 – Графіки з мінімальною вартістю за методами для збалансованих даних

Можливо експортувати отриманий план у Excel (рисунок 2.10), що полегшує подальшу роботу з результатами та їх включення у звіти або зовнішні системи.



	A	B	C	D	E	F
1	Dim1	Dim2	Dim3	Dim4	Value	
2	0	0	0	2	64	
3	0	1	1	0	6	
4	0	1	1	1	45	
5	0	2	0	1	21	
6	1	0	0	0	10	
7	1	1	0	0	20	
8	1	1	2	0	4	
9	2	1	0	0	22	
10	2	1	0	2	8	
11						

Рисунок 2.10 – Остаточний план перевезень, експортований до Excel

Таким чином, інтерфейс виконує не лише допоміжну роль, а й виконує функції повноцінної аналітичної панелі, що дозволяє як дослідження, так і використовувати продукт у прикладній логістиці.

Розроблений графічний інтерфейс охоплює всі етапи виконання завдання: від завантаження та налаштування перед аналізом і порівнянням результатів. Це робить програму не лише інструментом для виконання обчислень, а й зручним інструментом для вивчення поведінки алгоритму за різними параметрами та структурами даних.

2.6 Тестування на збалансованих даних

На другому та третьому етапах тестування проводилося автоматично за спеціально розробленою програмою (Додаток Б). У рамках тестування робота генетичного алгоритму оцінювалася на завданнях, у яких сума ресурсів і потреб на кожній осі співпадає. Такі завдання є класичною формулюванням збалансованого транспорту і дозволяють перевіряти як коректність реалізації, так і стабільність алгоритму для різних структур даних[18].

Для тестування програмної системи та оцінки стабільності алгоритму використовувалися задачі різних розмірів - від класичних двовимірних до узагальнених семивимірних. На початковому етапі реалізацію перевіряли за допомогою прикладів, запозичених із спеціалізованих підручників і посібників з транспортних завдань[19]. Результати, отримані програмою, повністю збіглися з еталонними розв'язками, які підтвердили коректність як класичних методів, так і структури генетичного алгоритму. Після цього було вирішено використати генератор задач (Додаток В), розроблений мною, який формує як збалансовані, так і незбалансовані формулювання довільної розмірності - як у двовимірному, так і в багатоіндексній формі. Такий підхід дозволив більш повно охопити потенційні сценарії, що зустрічаються в реальних бізнес-процесах, зокрема ті, де немає чіткого взаємозв'язку між попитом і пропозицією або існують додаткові шари логістичних зв'язків.

Для забезпечення порівнянності результатів у всіх тестах використовували єдиний набір параметрів генетичного алгоритму:

розмір популяції - 100 особин,

кількість поколінь - 100,

ймовірність мутації - 2%,

кількість островів - 4.

Ці параметри були обрані на основі попереднього аналізу для забезпечення стабільної роботи алгоритму. Хоча оптимальні параметри мали бути адаптовані до конкретного завдання, ті ж налаштування використовувалися для тестування, що показувало задовільні результати з задач усіх вимірів, навіть якщо це супроводжувалося надмірною кількістю поколінь для задач меншого масштабу. Початкова популяція на кожному острові формувалася за допомогою одного з наступних методів: метод північно-західного кута, метод мінімальних елементів, розв'язувач HiGHS та генератор випадкових даних[20]. Цей підхід дозволяв порівнювати ефективність різних стратегій ініціалізації в межах одного запуску алгоритму.

Під час виконання алгоритму були зібрані ключові показники: кінцеві витрати, кількість поколінь до стабілізації, час виконання та кількість прийнятних рішень у популяції. Крім того, після завершення кожного експерименту перевірялася відповідність обмеженням кожної осі, що дозволяло фіксувати випадки порушення умов завдання.

Для кожного запуску динаміка функції пристосованості за поколіннями візуалізувалася як для окремих островів, так і в сукупності. Це дозволило оцінити не лише кінцеву якість рішень, а й швидкість збіжності. Приклади таких графів наведені на рисунку 2.9. У всіх завданнях спостерігалося плавне навчання та різкі стрибки в точках обміну рішеннями між островами, що свідчить про правильну роботу моделі острова, операторів відбору, кросовера та мутації.

Аналіз фінальних рішень показав, що у всіх тестах можна отримати дійсний план без порушення меж. Водночас кінцева вартість, отримана за допомогою генетичного алгоритму, або збігалася з якістю (тобто була нижчою) за тими

рішеннями, які формувалися за допомогою окремих методів насіння. Таблиця 2.1 нижче ілюструє порівняння середніх загальних витрат для всіх прикладів, а також показує, який метод ініціалізації був найефективнішим у межах кожного острова.

Таблиця 2.1 – Порівняльний аналіз якості рішень

Характеристика	Метод північно-західного кута	Метод мінімального елементу	Промисловий розв'язувач HiGHS	Генетичний алгоритм
Середнє значення рішення	6900	3432	2573	2963
Середнє відставання від оптимуму, (од)	4328	860	0	390
Середнє відставання від оптимуму, (%)	168	33	0	15

Слід зазначити, що час виконання всіх обчислень коливався від однієї до трьох хвилин, залежно від розміру завдань, від двовимірного (Таблиця 2.2) до семивимірного (Таблиця 2.3).

Таблиця 2.2 – Середнє відставання від оптимуму при тестуванні двомірних задач

Методи розв’язку	Розмірність	Середнє відставання від оптимуму (од.)	Середнє відставання від оптимуму (%)	Найкраще рішення, од	Найгірше рішення,од.
Метод північно-західного кута	2d	32	1	7452	
Метод мінімаль-ного елементу		682	17	6269	
Промисловий розв'язувач HiGHS		0	0	5736	
Генетичний алгоритм		0	0	5736	5736

Це підтверджує застосування розробленої системи в реальних умовах, де важливі не лише точність, а й швидкість. Водночас адаптивна функція пристосованості, яка враховує штрафи за недонавантаження та перевантаження, не заважала ефективній збіжності алгоритму, а, навпаки, сприяла стабілізації результатів.

Таблиця 2.3 – Середнє відставання від оптимуму при тестуванні сімивимірних задач

Методи розв’язку	Розмірність	Середнє відставання від оптимуму (од.)	Середнє відставання від оптимуму (%)	Найкраще рішення (од.)	Найгірше рішення (од.)
Метод північно-західного кута	7d	4526	824	5065	
Метод мінімального елементу		1698	309	2247	
Промисловий розв'язувач HiGHS		0	0	549	
Генетичний алгоритм		849	155	549	2247

Таким чином, тестування на збалансованих даних підтвердило як коректність реалізації, так і високу ефективність запропонованого підходу. Алгоритм стійкий до розмірності задачі, стабільно знаходить прийнятні рішення, а завдяки вбудованому порівнянню з базовими методами дозволяє візуально оцінити його переваги.

2.7 Тестування на незбалансованих даних

На наступному етапі було проведено серію експериментів із використанням незбалансованих формулювань задачі транспорту, де сума пропозицій на одній або кількох осях не збігається зі сумою вимог. Такі ситуації часто виникають у

практичних логістичних питаннях через наявність надлишкових ресурсів, недонавантаження або нерівномірний розподіл попиту і пропозиції. Для роботи з такими випадками використовувалася та сама програмна реалізація генетичного алгоритму без необхідності змінювати структуру коду або вручну перемикати режими роботи[22].

Тестування проводилося на наборах задач тієї ж розмірності, що й у попередньому розділі - від двовимірних до семивимірних. Використовувалися ідентичні параметри алгоритму, за винятком збільшеної кількості поколінь: для підвищення ймовірності отримання якісних рішень параметр подвоїли і до 200 поколінь. Головна відмінність полягала у зміненому формулюванні функції пристосованості, яка враховувала штрафні значення за недостатню або надлишкову кількість для кожної осі. Це дозволило алгоритму ефективно розрізняти прийнятні та неприйнятні рішення, правильно керуючи процесом еволюції.

У ході експериментів фіксувалися значення фінальної цільової функції, кількість порушень обмежень, ступінь перерозподілу ресурсів, а також частка дозволених особин у популяції на кожному поколінні. Генетичний алгоритм продемонстрував стабільну поведінку: на початкових етапах еволюції переважали частково прийнятні рішення, але зі збільшенням кількості поколінь спостерігалося постійне зменшення їхньої частки та збільшення кількості особин, що відповідають обмеженням. Це підтверджує коректність механізму фільтрації та здатність алгоритму адаптуватися до складних вхідних даних.

Крім того, було проведено порівняння з класичними методами. Аналіз графіків (рисунок 2.11) показав, що при використанні методів північно-західного кута та мінімального елемента на початкових поколіннях генерувалися розв'язки, які не відповідали обмеженням, внаслідок чого спостерігалося збільшення значення цільової функції. Після появи прийнятних рішень вартість почала знижуватися. Варто зазначити, що навіть при використанні розв'язувача HiGHS (графік lp_method) у випадку незбалансованих задач на початкових етапах отримати оптимальні розв'язки неможливо, що підкреслює складність таких

формулювань і доцільність застосування адаптивних евристичних підходів.

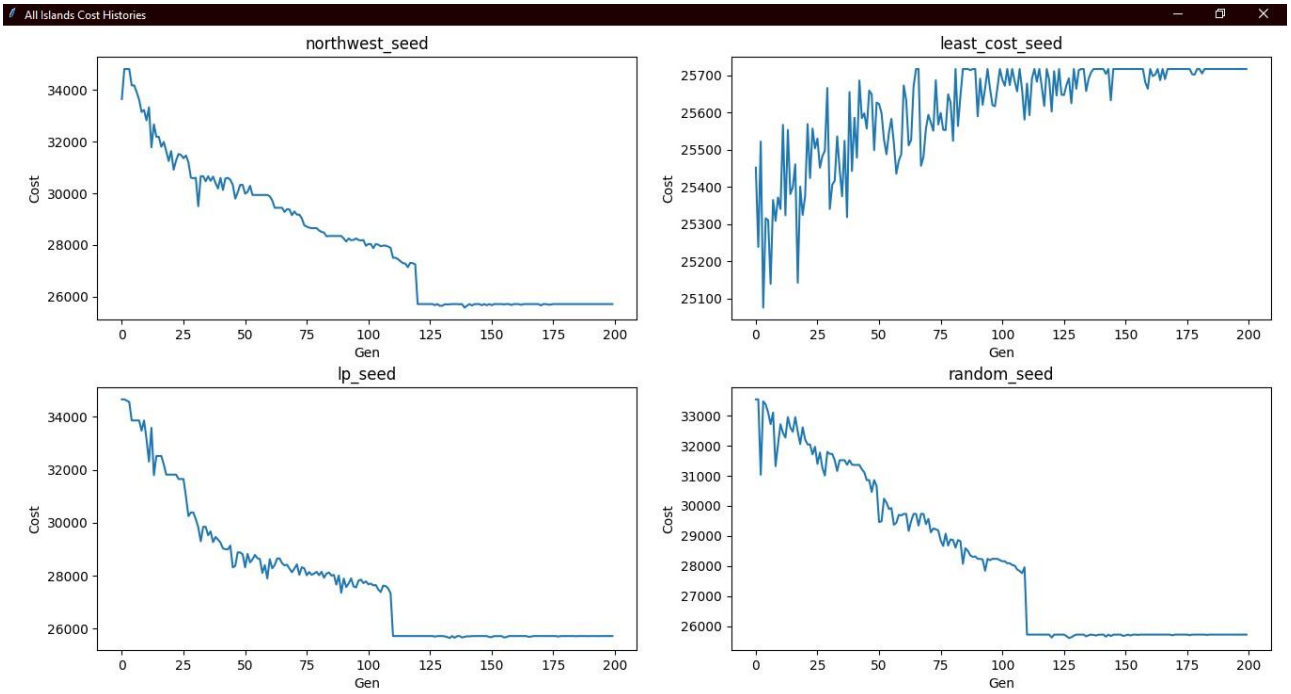


Рисунок 2.11 – Графіки мінімальної вартості за островами для незбалансованих даних

Остаточні плани, отримані на основі незбалансованих даних, відображали точний врахування логістичних обмежень. Відповідність вимогам перевірялася автоматично - відхилення по кожній осі відображалися в інтерфейсі користувача, що дозволяло негайно аналізувати «слабкі місця» отриманого плану (рисунок 2.12).

Візуалізація рішень, графіки збіжності та порівняння з результатами, отриманими базовими методами ініціалізації (див. Рис. 2.11-2.12) показали, що використання генетичного алгоритму залишається ефективним навіть у умовах незбалансованих систем. Незважаючи на можливу відсутність точного «ідеального» рішення, метод здатний адаптуватися до структури проблеми та створювати плани, які є економічно та логістично здійсненними з практичної точки зору.

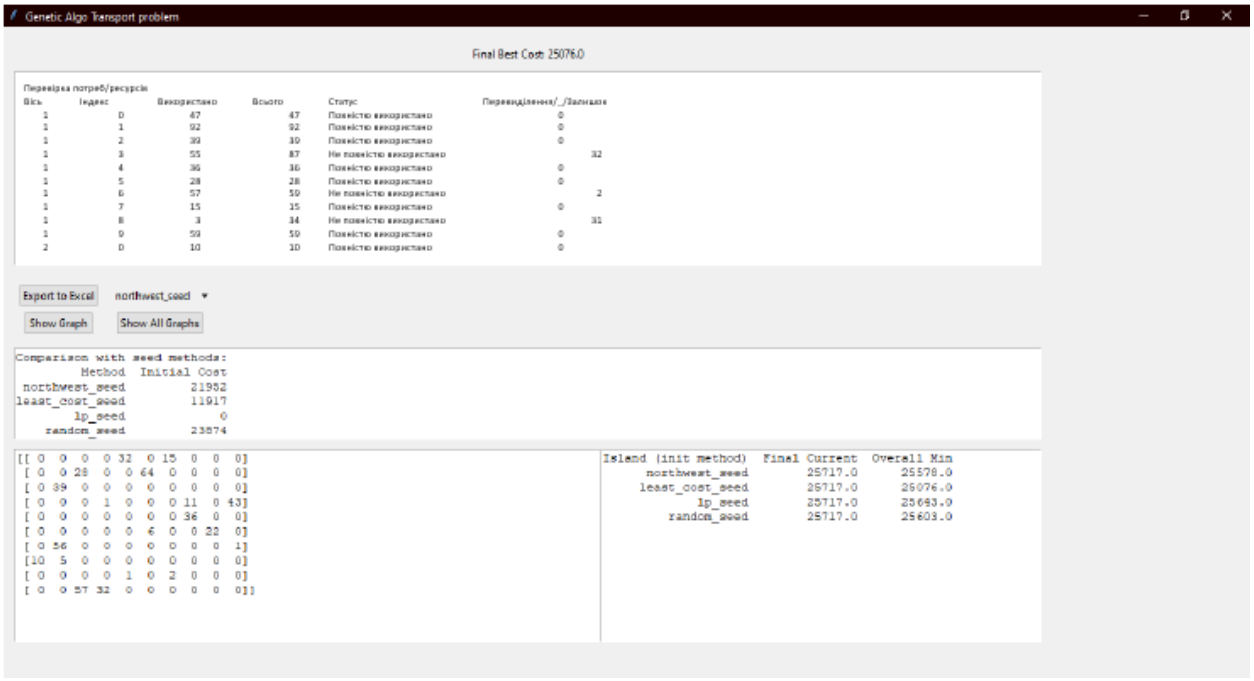


Рисунок 2.12 – Виведення алгоритму на незбалансованих даних

Експерименти підтверджують, що запропонований підхід застосовний не лише для строго формалізованих завдань, а й для більш реалістичних, «шумних» сценаріїв із надмірними або недостатніми ресурсами. Це розширює спектр застосувань розробленої системи та робить її придатною для завдань розподілу в умовах невизначеності.

Висновки до розділу:

Розробка програмного рішення для розв’язку транспортної задачі, включаючи її багатоіндексні модифікації, вимагала створення гнучкої та масштабованої архітектури, здатної адаптуватися до різних розмірів даних, методів розв'язання та критеріїв оцінки. Ключовою метою цієї реалізації було поєднання класичних алгоритмів і сучасних еволюційних підходів в одному програмному продукті, з можливістю порівняння їхньої якості та адаптації до практичних сценаріїв. Особливу увагу приділялося підтримці довільної розмірності завдання, а також зручності взаємодії користувача через графічний інтерфейс.

У ході реалізації було створено універсальну структуру, яка дозволяє вирішувати як класичні двовимірні, так і багатоіндексні транспортні задачі. Узагальнюючи модель, можна було звести обробку даних до однієї форми, що забезпечувало послідовну логіку для всіх методів розв'язання, що використовувалися в системі. Це не лише спростило підтримку коду, а й забезпечило стабільний результат, включно з експортом у таблиці, візуалізацією транспортних планів і побудовою порівняльних графіків.

ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи розроблено та досліджено універсальну програмну систему для розв’язання транспортних задач довільної розмірності, зокрема багатоіндексних і незбалансованих постановок, які є найбільш характерними для сучасних логістичних процесів.

У ході дослідження проведено аналіз класичних, евристичних і метаевристичних методів оптимізації, визначено їхні можливості та обмеження при застосуванні до задач великої розмірності та зі складною структурою обмежень. На цій основі обґрунтовано використання генетичного алгоритму з острівною моделлю як ефективного інструменту розв’язання задач такого типу, що забезпечує гнучкість, масштабованість і здатність працювати з неформалізованими даними.

Запропоновано універсальне представлення транспортної задачі, яке підтримує довільну кількість індексів і дозволяє описувати як класичні, так і узагальнені моделі. Розроблена структура забезпечує ефективне зберігання, обробку та сумісність із різними методами генерації та аналізу рішень.

Реалізовано генетичний алгоритм з урахуванням особливостей задач високої розмірності, включаючи коректне кодування рішень, побудову функції пристосованості з урахуванням обмежень, а також застосування операторів кросоверу та мутації, що забезпечують допустимість отриманих рішень. Для підвищення ефективності алгоритму розроблено систему формування початкової популяції, яка поєднує класичні методи (метод північно-західного кута, метод мінімального елемента), випадкову генерацію та використання спеціалізованих розв’язувачів.

Створено програмний засіб із графічним інтерфейсом користувача, що забезпечує постановку задач довільної розмірності, налаштування параметрів алгоритму, візуалізацію процесу оптимізації, аналіз результатів та їх експорт у зручному форматі. Реалізовані засоби візуалізації дозволяють відстежувати динаміку роботи алгоритму та структуру отриманих рішень.

Проведені експериментальні дослідження підтвердили коректність роботи розроблених алгоритмів, їхню стійкість та ефективність. Порівняння з класичними методами показало переваги запропонованого підходу при розв’язанні задач підвищеної складності, що виходять за межі стандартних моделей.

Отримані результати можуть бути використані при створенні систем підтримки прийняття рішень у сфері логістики та управління ресурсами, а також у навчальному процесі для вивчення методів оптимізації та еволюційних алгоритмів. Розроблена система має потенціал для подальшого розвитку, зокрема шляхом розширення функціональності, врахування додаткових критеріїв та інтеграції нових методів оптимізації.

Таким чином, поставлена мета роботи досягнута, а розроблене програмне рішення є ефективним інструментом для розв’язання широкого класу задач транспортної логістики.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1 Жалдак М.І., Триус Ю.В. Основи теорії і методів оптимізації : навчальний посібник. Черкаси : Брама-Україна, 2005. 608 с.

2 Сергиенко И.В., Гуляницкий Л.Ф., Сиренко С.И. Классификация прикладных методов комбинаторной оптимизации. *Кибернетика и системный анализ.* 2009. № 5. С. 71–83. https://doi.org/10.1007/s10559-009-9134-0

3 Гулаєва Н.М., Шило В.П., Глибовець М.М. Генетичні алгоритми як обчислювальні методи скінченновимірної оптимізації. *Кибернетика и системный анализ.* 2021. № 3. С. 5–14. https://doi.org/10.34229/2707-451X.21.3.1

4 Катренко А.В. Дослідження операцій. Львів : Магнолія 2006,350 с

5 Лавров Є. А., Перхун Л. П., Шендрик В. В. Математичні методи дослідження операцій. Суми : Сумський державний університет, 2017. 212 с.

6 Лісовиченко О. І., Калініна І. В. Використання генетичних алгоритмів в задачах оптимізації. *Адаптивні системи автоматичного управління.* 2015. Том 1, № 26. https://doi.org/10.20535/1560-8956.26.2015.45507

7 Гончарук К. С. Генетичні алгоритми в задачах синтезу маршрутів транспортних систем ГВС. *Адаптивні системи автоматичного управління.* 2011. Том 1, № 18.https://doi.org/10.20535/1560-8956.18.2011.33475

8 Глибовець М.М., Гулаєва Н.М. Еволюційні алгоритми : підручник. Київ : НаУКМА, 2013. 828 с

9 David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company. 1989, 412pp

10 Зайченко Ю. П. Дослідження операцій : Підручник. К. : Слово, 2006. 816с.

11 Meng Q., Wu J., Ellisy J., Kennedy P. J. Dynamic island model based on spectral clustering in genetic algorithm. arXiv. 2018. URL: https://arxiv.org/abs/1801.01620 (Дата звернення: 23.04.2026).

12 McKinney W. Python for data analysis. 3rd ed. Sebastopol : O'Reilly Media, 2022. 550 p.

13 Harris C. R., Millman K. J., van der Walt S. J. et al. Array programming with NumPy. Nature. 2020. Vol. 585. P. 357–362.

14 Gleixner A. M., Steffy D. E., Wright S. J. The HiGHS linear optimization software. arXiv. 2020. URL: https://arxiv.org/abs/2002.04065 (Дата звернення: 23.04.2026).

15 Hunter J. D. Matplotlib: a 2D graphics environment. *Computing in Science & Engineering*. 2007. Vol. 9(3). P. 90–95.

16 Simon D. Evolutionary optimization algorithms. Hoboken : Wiley, 2013. 512 p.

17 Taha H. A. Operations research: an introduction. 10th ed. Boston : Pearson, 2017. 848 p.

18 Vicary J. Implementing the HiGHS linear constraint solver in Rust. Cambridge : University of Cambridge, 2022. URL: https://github.com/jlcv/highs-rs (Дата звернення: 20.05.2026).

19 Simoncini D., Collard P., Verel S., Clergue M. From cells to islands: an unified model of cellular parallel genetic algorithms. arXiv. 2008. URL: https://arxiv.org/abs/0803.4248 (Дата звернення: 20.05.2026).

20 Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 "Комп'ютерні науки". Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

21 ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

22 ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

23 ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).

ДОДАТОК А

Лістинг коду розв’язку транспортної задачі

```
import json
import random
import threading
import time
import numpy as np
from copy import deepcopy
from scipy.optimize import linprog
import pandas as pd
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import os
import sys
sys.stdout.reconfigure(encoding="utf-8")
# DATA LOADING
def read_data_from_file(path):
    with open(path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    constraints = [np.array(m) for m in data['margins']]
    costs = np.array(data['costs'])
    return constraints, costs
# SEED METHODS
def northwest_seed(constraints, costs):
    rem = [c.copy() for c in constraints]
    plan = np.zeros(costs.shape, int)
    shape = costs.shape
    idx = [0] * costs.ndim
    while all(idx[d] < shape[d] for d in range(costs.ndim)):
        avail = min(rem[d][idx[d]] for d in range(costs.ndim))
        plan[tuple(idx)] = avail
        for d in range(costs.ndim):
            rem[d][idx[d]] -= avail
        for d in range(costs.ndim):
            if rem[d][idx[d]] == 0:
                idx[d] += 1
                break
    return plan
def least_cost_seed(constraints, costs):
    rem = [c.copy() for c in constraints]
    plan = np.zeros(costs.shape, int)
    cells = list(np.ndindex(costs.shape))
```

```
    cells.sort(key=lambda i: costs[i])
    for i in cells:
        avail = min(rem[d][i[d]] for d in range(costs.ndim))
        if avail > 0:
            plan[i] = avail
            for d in range(costs.ndim):
                rem[d][i[d]] -= avail
    return plan
def random_seed(constraints, costs):
    rem = [c.copy() for c in constraints]
    plan = np.zeros(costs.shape, int)
    cells = list(np.ndindex(costs.shape))
    random.shuffle(cells)
    for i in cells:
        avail = min(rem[d][i[d]] for d in range(costs.ndim))
        if avail > 0:
            x = random.randint(0, avail)
            plan[i] = x
            for d in range(costs.ndim):
                rem[d][i[d]] -= x
    return plan
# GUI
def run_gui():
    root = tk.Tk()
    root.title("Genetic Algo Transport problem")
    root.state("zoomed")
    constraints, costs = None, None
    stop_event = threading.Event()
    # LOAD
    def load_file():
        nonlocal constraints, costs
        fn = filedialog.askopenfilename(filetypes=[('JSON', '*.json')])
        if not fn:
            return
        constraints, costs = read_data_from_file(fn)
        messagebox.showinfo("Loaded", f"File loaded: {fn}")
    #RUN
    def start_algorithm():
        nonlocal constraints, costs
        if constraints is None or costs is None:
            messagebox.showerror("Error", "Load data first")
            return
        messagebox.showinfo("Info", "Algorithm started (simplified GUI version)")
    # UI
    ttk.Button(root, text="Load JSON", command=load_file).pack()
```

```
    ttk.Button(root, text="Run", command=start_algorithm).pack()
    root.mainloop()
if __name__ == '__main__':
    run_gui()
```

ДОДАТОК Б

Лістинг коду для автоматичного розв’язку транспортних задач

```
import pandas as pd
import numpy as np
import random
from Generator import generate_instance_nd
from Transport_Problem import (
   genetic_algorithm_islands,
   northwest_seed, least_cost_seed, HiGHS_seed, random_seed
)
import sys
sys.stdout.reconfigure(encoding="utf-8")
# Фіксуємо seed для відтворюваності
random.seed(42)
np.random.seed(42)
# Параметри експериментів
EXPERIMENTS = [
   {'dims': [100, 100], 'num_tasks': 10, 'runs': 10},
   {'dims': [3, 3, 3, 3, 3, 3, 3], 'num_tasks': 10, 'runs': 10},
   {'dims': [10, 10, 10, 10], 'num_tasks': 10, 'runs': 10},
   {'dims': [3, 3], 'num_tasks': 10, 'runs': 10},
   {'dims': [5, 5], 'num_tasks': 10, 'runs': 10},
   {'dims': [7, 7], 'num_tasks': 10, 'runs': 10},
   {'dims': [3, 3, 3], 'num_tasks': 10, 'runs': 10},
   {'dims': [5, 5, 5], 'num_tasks': 10, 'runs': 10},
   {'dims': [7, 7, 7], 'num_tasks': 10, 'runs': 10},
   {'dims': [3, 3, 3, 3], 'num_tasks': 10, 'runs': 10},
   {'dims': [5, 5, 5, 5], 'num_tasks': 10, 'runs': 10},
]
POP_SIZE = 100
GENERATIONS = 100
ISLANDS = 4
SEED_RATIO = 0.25
MUTATION_RATE = 0.02
SEED_METHODS = [
   ('Northwest', northwest_seed),
   ('LeastCost', least_cost_seed),
   ('LP', HiGHS_seed),
   ('Random', random_seed)
]
def run_experiment(exp, balance_margins=False):
   dims = exp['dims']
   num_tasks = exp['num_tasks']
   runs = exp['runs']
   records = []
   for task_id in range(1, num_tasks + 1):
      #  balance_margins передається правильно
      instance = generate_instance_nd(
```

```
    dims,
    balance_margins=balance_margins,
    normalize_costs_flag=False
)
constraints = [np.array(m) for m in instance['margins']]
costs = np.array(instance['costs'])
# --- Початкові методи ---
for name, method in SEED_METHODS:
    start = time.time()
    try:
        plan = method(constraints, costs)
        if plan.shape != costs.shape:
            raise ValueError(f"{name}: неправильна форма плану")
        cost_val = float(np.sum(plan * costs))
    except Exception as e:
        print(f"[ERROR] {name} failed on task {task_id}: {e}")
        cost_val = np.nan
    t = time.time() - start
    records.append({
        'dims': tuple(dims),
        'task_id': task_id,
        'method': name,
        'run_id': 0,
        'best_cost': cost_val,
        'time_sec': t
    })
# --- Генетичний алгоритм ---
for run_id in range(1, runs + 1):
    start = time.time()
    try:
        _, best_cost, _ = genetic_algorithm_islands(
            constraints,
            costs,
            pop_size=POP_SIZE,
            generations=GENERATIONS,
            islands=ISLANDS,
            seed_ratio=SEED_RATIO,
            mutation_rate=MUTATION_RATE
        )
        best_cost = float(best_cost)
    except Exception as e:
        print(f"[ERROR] GA failed on task {task_id}, run {run_id}: {e}")
        best_cost = np.nan
    t = time.time() - start
    records.append({
        'dims': tuple(dims),
        'task_id': task_id,
        'method': 'GA',
        'run_id': run_id,
```

```
            'best_cost': best_cost,
            'time_sec': t
        })
    print(f"Completed dims={dims}, task {task_id}/{num_tasks}")
  return pd.DataFrame(records)
def main(balance_margins=False):
  all_dfs = []
  for exp in EXPERIMENTS:
    print(f"Running experiment for dims={exp['dims']}")
    df = run_experiment(exp, balance_margins)
    all_dfs.append(df)
  full_df = pd.concat(all_dfs, ignore_index=True)
  output_file = 'multi_test_results.xlsx'
  full_df.to_excel(output_file, index=False)
  print(f"All experiments done. Results saved to {output_file}")
if __name__ == '__main__':
  main(balance_margins=False)
```

ДОДАТОК В

Лістинг коду для генерації транспортних задач

```
import json
import numpy as np
import os
import random
import sys
sys.stdout.reconfigure(encoding="utf-8")
def generate_margins_2d(num_suppliers, num_consumers, total=None, balance_margins=True):
    if balance_margins:
        if total is None:
            total = random.randint(50, 200)
        k = num_suppliers + num_consumers - 1
        if k >= total:
            raise ValueError("Занадто багато постачальників/споживачів для заданого total")
        cuts = sorted(random.sample(range(1, total), k))
        parts = [cuts[0]] + [cuts[i] - cuts[i - 1] for i in range(1, len(cuts))] + [total - cuts[-1]]
        supplies = parts[:num_suppliers]
        demands = parts[num_suppliers:]
    else:
        supplies = [random.randint(0, 100) for _ in range(num_suppliers)]
        demands = [random.randint(0, 100) for _ in range(num_consumers)]
    return supplies, demands
def generate_costs_2d(num_suppliers, num_consumers, low=1, high=100):
    return np.random.randint(low, high + 1, size=(num_suppliers, num_consumers)).tolist()
def normalize_costs(costs):
    arr = np.array(costs, dtype=float)
    minc, maxc = arr.min(), arr.max()
    if maxc == minc:
        return arr.tolist(), (minc, maxc)
    norm = (arr - minc) / (maxc - minc)
    return norm.tolist(), (minc, maxc)
def generate_instance_2d(num_suppliers, num_consumers,
                         total=None, balance_margins=True,
                         cost_low=1, cost_high=100,
                         normalize_costs_flag=False,
                         output_path=None):
    supplies, demands = generate_margins_2d(num_suppliers, num_consumers, total,
balance_margins)
    costs = generate_costs_2d(num_suppliers, num_consumers, cost_low, cost_high)
    norm_params = None
    if normalize_costs_flag:
        costs, norm_params = normalize_costs(costs)
    instance = {
        "margins": [supplies, demands],
        "costs": costs,
        "balanced_margins": balance_margins,
```

```
        "normalized_costs": normalize_costs_flag,
        "normalization_params": norm_params
    }
    if output_path:
        dir_name = os.path.dirname(output_path)
        if dir_name:
            os.makedirs(dir_name, exist_ok=True)
        with open(output_path, 'w', encoding='utf-8') as f:
            json.dump(instance, f, ensure_ascii=False, indent=2)
    return instance
def generate_margins_nd(dim_sizes, total=None, balance_margins=True):
    margins = []
    if balance_margins:
        if total is None:
            total = random.randint(50, 200)
        for size in dim_sizes:
            if size > total:
                raise ValueError("Розмір виміру не може бути більшим за total")
            cuts = sorted(random.sample(range(1, total), size - 1))
            parts = [cuts[0]] + [cuts[i] - cuts[i - 1] for i in range(1, len(cuts))] + [total - cuts[-1]]
            margins.append(parts)
    else:
        for size in dim_sizes:
            margins.append([random.randint(0, 100) for _ in range(size)])
    return margins
def generate_costs_nd(dim_sizes, low=1, high=100):
    return np.random.randint(low, high + 1, size=tuple(dim_sizes)).tolist()
def generate_instance_nd(dim_sizes,
                         total=None, balance_margins=True,
                         cost_low=1, cost_high=100,
                         normalize_costs_flag=False,
                         output_path=None):
    margins = generate_margins_nd(dim_sizes, total, balance_margins)
    costs = generate_costs_nd(dim_sizes, cost_low, cost_high)
    norm_params = None
    if normalize_costs_flag:
        arr = np.array(costs, dtype=float)
        minc, maxc = arr.min(), arr.max()
        if maxc != minc:
            arr = (arr - minc) / (maxc - minc)
        costs = arr.tolist()
        norm_params = (float(minc), float(maxc))
    instance = {
        "margins": margins,
        "costs": costs,
        "balanced_margins": balance_margins,
        "normalized_costs": normalize_costs_flag,
        "normalization_params": norm_params
    }
```

```
    if output_path:
        dir_name = os.path.dirname(output_path)
        if dir_name:
            os.makedirs(dir_name, exist_ok=True)
        with open(output_path, 'w', encoding='utf-8') as f:
            json.dump(instance, f, ensure_ascii=False, indent=2)
    return instance
# (опціонально) для відтворюваності
def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
if __name__ == "__main__":
    set_seed(42)
    generate_instance_nd(
        [4, 4, 4, 4],
        total=500,
        balance_margins=True,
        normalize_costs_flag=False,
        output_path="./data/4d_bal.json"
    )
```