МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗВ'ЯЗАННЯ ЗАДАЧІ ПРО ЗАВАНТАЖЕННЯ ТРАНСПОРТУ З ВХІДНИМИ ДЕТЕРМІНОВАНИМИ ДАНИМИ»**

Виконав: студент 2 курсу, групи 8.1112-з
спеціальності 111 математика
(шифр і назва спеціальності)
освітньої програми математика
(назва освітньої програми)
А.В. Омельчук
(ініціали та прізвище)
Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Ткаченко І.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра фундаментальної та прикладної математики

Рівень вищої освіти магістр

Спеціальність 111 математика
(шифр і назва)

Освітня програма математика

ЗАТВЕРДЖУЮ

Завідувач кафедри фундаментальної та прикладної математики, д.т.н., професор

_____ Гребенюк С.М.
(підпис)

« ____ » _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Омельчуку Андрію Вікторовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розв'язання задачі про завантаження транспорту з вхідними детермінованими даними

керівник роботи (проекту) Ткаченко Ірина Григорівна, к.ф.-м.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 643-с

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розв'язання задачі про завантаження транспорту з вхідними детермінованими даними.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 15.05.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	20.05.2023	виконано
2.	Збір вихідних даних.	25.05.2023	виконано
3.	Обробка методичних та теоретичних	25.06.2023	виконано
	джерел.		
4.	Розробка першого розділу.	15.08.2023	виконано
5.	Розробка другого розділу.	29.09.2023	виконано
6.	Оформлення та нормоконтроль	листопад 2023	виконано
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи магістра.	12.12.2023	

Студент _____ А.В. Омельчук
(підпис) (ініціали та прізвище)

Керівник роботи _____ І.Г. Ткаченко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розв’язання задачі про завантаження транспорту з вхідними детермінованими даними»: 64 с., 16 рис., 5 табл., 16 джерел.

ЖАДІБНИЙ МЕТОД, ЗАДАЧА ПРО ЗАВАНТАЖЕННЯ, КРИТЕРІЇ ОПТИМАЛЬНОСТІ, МЕТОД ДИНАМІЧНОГО ПРОГРАМУВАННЯ, МЕТОД ПЕРЕБОРУ ДАНИХ, МОДЕЛІ ДИНАМІЧНОГО ПРОГРАМУВАННЯ, СТАТИСТИЧНА НЕВИЗНАЧЕНІСТЬ, JAVA, MATLAB

Мета дослідження – розробка програмного забезпечення для розв`язання задачі завантаження транспорту та дослідження впливу статистичної невизначеності на оптимальний розв'язок задачі.

Об’єкт дослідження – моделі динамічного програмування.

Предмет дослідження – задача про завантаження транспорту.

Методи дослідження – метод повного перебору, жадібний метод та метод динамічного програмування.

У кваліфікаційній роботі було отримано розв’язок детермінованої задачі про завантаження транспорту методом повного перебору та методом динамічного програмування. Розроблено програмне забезпечення для розв’язання задачі про завантаження транспорту методами повного перебору, жадібним методом та динамічного програмування в умовах статистичної невизначеності. Запропоновано різні варіанти критеріїв оптимальності. Отримано та проаналізовано розв’язок задачі з вхідними параметрами, розподіленими за нормальним законом.

SUMMARY

Master’s Qualification Thesis «Solving the problem of loading transport with input deterministic data»: 64 pages, 16 figures, 5 tables, 16 sources.

GREEDY METHOD, LOADING PROBLEM, OPTIMALITY CRITERIA, DYNAMIC PROGRAMMING METHOD, DATA ANALYSIS METHOD, DYNAMIC PROGRAMMING MODELS, STATISTICAL UNCERTAINTY, JAVA, MATLAB

The purpose of the research is to develop software for solving the problem of loading transport and to study the influence of statistical uncertainty on the optimal solution of the problem.

The object of the study is dynamic programming models.

The subject of the research is the task of loading the vehicle.

The method of research are the exhaustive search method, the greedy method, and the dynamic programming method.

In the qualifying paper, a solution to the deterministic problem of vehicle loading was obtained by the method of complete enumeration and the method of dynamic programming. Software has been developed for solving the problem of loading the transport by the methods of full enumeration, the greedy method and dynamic programming under conditions of statistical uncertainty. Different variants of optimality criteria are proposed. The solution of the problem with input parameters distributed according to the normal law was obtained and analyzed.

ЗМІСТ

ВСТУП

З часом для оптимізації використовується все більше різних методів, при чому їх складність зростає. Задачі дискретної оптимізації в реальному світі дуже складні. Сучасні методи оптимізації потребують вирішення реальних проблем за допомогою людини. Потрібні нові математичні моделі та методи, які б дозволили врахувати наявність кількох критеріїв і які б могли вести глобальний пошук найкращого з них. Слід віддавати перевагу методам, які полегшують процес знаходження розв'язку задачі.

Задача про літак – це NP -ідеальна комбінаторна оптимальна задача. Свою назву вона отримала через наступне: своєю метою вона мала якомога більше вантажу завантажити на літак, враховуючи його обмежену грузовантажність.

У загальному вигляді задачу можна подати так: із заданої множини елементів з властивостями «ціна» та «вага» необхідно вибрати той, який має найбільшу ціну, зберігаючи обмеження ваги.

Випадкова (статистична) похибка вимірювання – це компонент похибки вимірювання, який змінюється випадковим чином при повторних вимірюваннях тієї самої величини.

Вибір у статистично сумнівних ситуаціях робиться не тільки при оцінці вартості, а й при аналізі об'єктів. Ключовим або центральним припущенням для формалізації розв'язку задач динамічного програмування є припущення про статистичний характер експериментальних даних.

Невизначеність у статистичних питаннях має «дворівневий» характер. Дані, що спостерігаються, відповідають певному випадковому розподілу, і невизначеність, пов'язана з цим розподілом, створює «перший поверх». Існує інша невизначеність щодо розподілу деяких запропонованих експериментальних даних. Остання невизначеність повинна бути визнана шляхом вибору з набору альтернативних розподілів. Алгоритм такого виділення самого розподілу (або значень окремих його ознак) називається статистичним методом.

Прибуток у реальному житті не обов’язково повинен мати конкретне значення, яке може змінюватися залежно від багатьох умов. Він змінюється, коли кількість зменшується або збільшується (від курсу долара, податків, мит тощо), необхідно знати, чи змінюється товар у міру зміни середовища. Тому важливо розглядати проблему в умовах статистичної невизначеності.

Дана робота була представлена на конкурсах:

– студентських наукових робіт з галузей знань і спеціальностей у 2022/2023 н.р. спеціальності «Математика» і був нагороджений дипломом I ступеня;

– студентських наукових робіт Запорізького національного університету;

– роботі Чотирнадцятої Всеукраїнської, двадцять першої регіональної наукової конференції молодих дослідників «Актуальні проблеми математики та інформатики»;

– обласному конкурсі для обдарованої молоді у галузі наук Запорізької обласної державної адміністрації.

Робота опублікована в збірці тез доповідей Чотирнадцятої Всеукраїнської, двадцять першої регіональної наукової конференції молодих дослідників «Актуальні проблеми математики та інформатики» Запорізького національного університету 27-28 квітня 2023 р. УДК 519.213.1.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Динамічне програмування

У процесі розвитку, як і в економічних умовах, виникає необхідність вдосконалення господарських структур на всіх підприємствах. У даний час питанням організації та управління приділяється велика увага, що зумовлює необхідність комплексної розробки запропонованих процесів з точки зору їх структури та організації. Динамічне програмування (ДП) є одним із найефективніших методів розв’язання подібних задач, що пояснює важливість даної роботи.

Методи ДП використовуються в різноманітних наукових дослідженнях:

– в алгоритмах розпізнавання мови та зображень;

– при обробці великих масивів даних у соціології та економічній діяльності;

– для постановки плану розвитку виробничої бази промисловості та будівництва;

– для виділення капіталу, позики;

– при розробці правил управління попитом і запасами;

– при створенні плану-графіку поточних і капітальних ремонтів і замін обладнання;

– при знаходженні найкоротшої відстані в транспортній мережі;

– при підготовці робочої сили;

– при знаходженні оптимального плану раціонального завантаження транспортних засобів, оптимального управління транспортом

– при побудові послідовності включення об’єктів в будівельний рух тощо.

ДП зазвичай використовується для оптимізаційних і комбінаторних задач. Прикладами задач оптимізації є задача оптимального розподілу ресурсів, задача завантаження човнів (мішків), задача оснащення обладнання, задача

знаходження послідовності максимального зростання, задача порядку множення матриці (мінімізація числа скалярних множень), задача вибору траєкторії, задача конкуренції, задача управління запасами [1].

Розвиток теорії динамічного програмування відбувся в 1950-1953 рр. завдяки роботі Р. Беллмана та його однодумців. Першими проблемами, які призвели до створення методу бухгалтерського обліку, були динамічні проблеми управління рухом запасів на виробництвах [2].

ДП визначає оптимальний розв'язок n-вимірної задачі, розділяючи її на n кроків, кожен з яких є підпроблемою щодо однієї змінної. Обчислювальна перевага цього підходу полягає в тому, що ми маємо справу з розв'язанням одновимірних, а не багатовимірних задач оптимізації. Основним принципом ДП, на якому базується декомпозиція задачі на кроки, є оптимальність. Оскільки природа кожного кроку залежить від конкретної проблеми оптимізації, ДП безпосередньо не надає обчислювальних алгоритмів для кожного кроку. Обчислювальні аспекти розв'язання оптимізаційних задач проектуються та реалізуються окремо на кожному рівні (що, звичайно, не виключає використання єдиного алгоритму для всіх етапів) [3].

Покажемо застосування методу ДП на прикладі розв'язування задачі про літак. Проблема розв'язання таких задач актуальна та досить складна з точки зору застосування в реальному житті. Треба знайти найкращий спосіб завантаження продукції в транспорт (літак, поїзд, корабель тощо) або на склад. Задача про завантаження та її модифікації часто виникають в економічних, криптографічних, лінгвістичних, логістичних сферах.

Ця задача може бути розв'язана точним, наближеним або стохастичним методами. Точні методи включають повне перерахування та метод поширення та посилання. Група наближених методів представлена різновидами методів ДП. Стохастичні методи є жадібним і генетичним алгоритмами [4].

Єдиною проблемою є одна із задач максимізації, метою якої є визначення того, скільки цінностей можна помістити в мішок, коли обчислюється загальний об'єм (або вага) усіх об'єктів, які поміщаються в нього. У літературі можна зустріти обґрунтування та аналіз кількох методів розв'язування даної задачі,

серед яких найбільш поширеними є точний та зв'язаний метод, метод ДП та евристичний метод [5].

Таким чином, задача про літак (рюкзак) є інтегральною задачею лінійного програмування з булевими змінними. ЇЇ розв'язання відбувається в деяких підмножинах 2^n комбінацій, утворених різними розв'язками змінних x_i, $i = \overline{1; n}$. Відомо, що задача є *NP*-повною, тобто теоретично повного розв'язку не існує. Серед експоненціальних алгоритмів найбільш прийнятними є ті, що реалізують схему розгалуження та обмеження, метод ДП, адитивний алгоритм Балаша, а також інші розроблені алгоритми. Незважаючи на те, що ці алгоритми оцінюються за обчислювальною складністю, вони все ще не мають рейтингів практичного вирішення проблем, рейтингів середньої продуктивності алгоритму, що важливо для їх застосування. У [6] представлено планування та експериментальне дослідження трьох алгоритмів розв'язання задачі відображення зображення. Два з них – метод розгалуження та ДП – дозволяють отримати точні розв'язки, третій – евристичний метод – забезпечує наближені розв'язки задачі. Наведено порівняльну характеристику алгоритмів, на якій базуються рекомендації щодо їх використання.

У [7] розглянуто алгоритм точного розв'язку задачі про рюкзак (літак) за допомогою ДП. Основною особливістю цього алгоритму є залежність лінійної часової складності алгоритму від розмірності задачі. Вивчення алгоритму зазначеного методу є важливим, оскільки його швидкість визначає стабільність системи шифрування рюкзака.

Алгоритми мурашиних колоній обіцяють нові методи оптимізації, засновані на моделях поведінки цих колоній. Мурашину колонію можна розглядати як багатоагентну систему, де кожен агент працює за дуже простими правилами. На відміну від майже примітивної поведінки агентів, поведінка всієї системи виявляється напрочуд розумною.

Задачі комбінаторної оптимізації, такі як задача комівояжера, задача про присвоєння квадратів, задача кінотеатру, задача календарного планування, задача розфарбовування графа та задача рюкзака, отримали назву задачі мурашиних алгоритмів [8].

У роботі [8] було розглянуто формулювання класичної задачі сумки, для розв’язання цієї проблеми використано метод ідеального пошуку та два стохастичні методи: жадібний та генетичний алгоритми. Було проведено порівняння розглянутих методів. Також у цій роботі, як приклад, розв’язується класична задача сумки та практична задача пошуку найкращого транспорту.

Для знаходження точного розв’язку треба використовувати точний метод. Оскільки метод повного пошуку займає багато часу, необхідно обмежити розмір сумки та кількість речей, що використовуються. Якщо перебрати весь фіксований набір з n елементів, то розв’язок буде знайдено зі складністю не менше, ніж $O(2n)$.

Стохастичні методи працюють набагато швидше, ніж точні, але не можуть знайти точний розв’язок. Генетичний алгоритм є найшвидшим з алгоритмів. Недоліком цього алгоритму є те, що в деяких ситуаціях він знаходить кінцеве розташування замість глобального. Алгоритм може бути припинений не тільки при досягненні оптимального розв’язку, але і за наступних умов:

– пропущено максимальну кількість ітерацій;

– минув максимальний час, заданий для виконання алгоритму;

– при переході до наступного покоління не відбувається істотних змін.

1.2 Що таке Java

Java – об’єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (пізніше придбана Oracle) [9]. Офіційна дата випуску – 23 травня 1995 р. За офіційною заявою виробника Java використовується:

- 97% робочих столів;
- 89% настільних комп'ютерів у США;
- 3 мільярди мобільних пристроїв;
- 125 мільйонів телевізорів.

Особливе місце тут займає Java Enterprise Edition. Реалізація Java призначена для корпоративного програмного забезпечення, багатопотокових розподілених систем, додатків корпоративного рівня: різноманітних банківських систем, систем планування ресурсів підприємства (ERP), веб-сервісів тощо.

Java має кілька функцій, які роблять її популярною серед розробників:

- портативність: Java є стек-незалежною мовою через використання JVM;
- об'єктно-орієнтований підхід;
- індикатор бібліотеки;
- безпека: Java розроблено з урахуванням безпеки. Ваша пісочниця обмежує доступ до обчислювальних ресурсів, допомагаючи запобігати небезпечним діям і підтримувати роботу програм;
- автоматичне керування пам'яттю: механізм збирання сміття в інструментах Java, який дозволяє автоматично звільняти пам’ять від використаних об’єктів, які не використовуються в програмі.

Java, як і раніше, залишається популярною мовою та знаходиться в стадії активного розвитку. Протягом багатьох років з’являлися нові версії Java, включаючи Java 8, Java 9, Java 10 та новіші. Вони пропонують нові функції, покращення продуктивності та розширену бібліотеку правил. Java – потужна, портативна та надійна мова програмування, яка знаходить застосування в різних сферах. Його об’єктно-орієнтований підхід, багата бібліотека та безпека роблять його найкращим вибором для багатьох розробників. Завдяки активній спільноті розробників і постійному розвитку мови Java продовжує залишатися одним із провідних інструментів програмування у світі [10].

Тепер, переходячи до конкретних переваг Java, розглянемо кожну з них докладніше. Однією з найбільших переваг мови є її багата бібліотека. З одного боку, величезна бібліотека є однією з перешкод для вивчення мови, що робить бар’єр входу високим. Однак через високий бар’єр для входу популярність мови впала, а попит на мову не змінився. Це вигідно нам у вигляді високої зарплати для Java-програмістів, відносно невеликої конкуренції, можливості безкоштовного навчання Java від зацікавлених компаній, а також додаткової

сертифікації та роботи в тих же компаніях. Багата бібліотека підсумовує тривіальну частину роботи програміста: пошук готового рішення, яке звільняє програміста від рутини та дозволяє йому серйозніше інвестувати в творчий аспект і заощадити час і зусилля.

У Java байт-код використовується, коли певний код (байт-код) створюється з вихідного тексту програми. Для виконання цього алгоритму коду використовується спеціальний інтерпретатор. Він послідовно перетворює інструкцію байт-коду у відповідну апаратну інструкцію та виконує її, а потім транслює наступну інструкцію байт-коду. Крім того, байт-код оптимізовано для платформи, щоб його виконання займало найменшу кількість часу та ресурсів. При цьому на інших платформах програма працює коректно, головне, щоб був встановлений інтерпретатор цього байт-коду. Java ніколи не була популярним інструментом для розробки настільних додатків, але вона процвітала в сегменті мобільного ринку, який останнім часом стрімко зростає. Платформа Android повністю побудована на основі Java, і наразі пристрої Android є найпопулярнішими на ринку. Це майстерність новизни. Зменшена версія мови та віртуальної машини, відома як Java ME, широко використовується в багатьох так званих функціональних телефонах, яких мільйони по всьому світу. Додайте все це, і домінування Java буде приголомшливим.

Java, яка раніше називалася «Дуб», призначалася для телевізорів, де хотіла домінувати Sun. Точно дотримуватися плану було неможливо, але Java вдалося знайти затишне місце в кімнаті. Стандарт Blu-Ray базується на Java, і кожен, хто бажає додати до Blu-Ray додатковий вміст, повинен буде використовувати компілятор Javac. Диски Blu-ray – це не просто необроблені фільми. Код Java можна використовувати для зміни/додавання додаткових функцій та інтерактивності. Диски Blu-ray – це комбінація стисненого відео та байт-коду Java [11].

Java не була першою мовою, якою були написані кросплатформні програми, але вона стала найпопулярнішою. Це не означає повної сумісності між платформами – відсутність бібліотек або несумісних версій бібліотек просто поховає ваш код. Неможливо завантажити код настільної програми,

скомпільований у JRE 1.7, і запустити його на телефоні Java ME. Дива не станеться. Sun і тепер Oracle намагаються створити кросплатформенну платформу. Якщо код не працює, зазвичай очевидно, у чому проблема. Якщо ви використовуєте правильні версії Java і маєте достатньо пам’яті, код працюватиме. Розробники Java можуть розробити програму на своєму комп’ютері, а потім розгорнути її на цільовій платформі, будь то телефон або сервер. Якщо компілятор має доступ до необхідних бібліотек, код працюватиме.

Багато було сказано про переваги Java, а тепер про недоліки.

За деякими даними, час виконання одних і тих же завдань у продуктах цієї мови в два з половиною рази повільніше, ніж у тому ж C. Однак, незважаючи на те, що переваги мови Java переважують її недоліки, вона все одно є однією з більш відомих. Наприклад, подвійна рекурсія в Java виявилася в 1,5 рази повільнішою при обчисленні чисел Фібоначчі.

Аналізуючи всі переваги та недоліки мови програмування Java, можна зробити висновок, що це універсальний, поширений продукт, який має багато переваг перед іншими мовами програмування [12].

2 ЗАДАЧА ПРО ЗАВАНТАЖЕННЯ ТРАНСПОРТУ

2.1 Постановка задачі

Транспорт завантажується n предметами різних типів. Кожен предмет типу j дає прибуток c_j одиниць і важить a_j тон. Вантажопідйомність транспорту приймемо за b тон. Необхідно обрати предмети, навантаження яких дозволить отримати максимальний дохід без перевищення вантажопідйомності транспорту.

Математична постановка задачі має наступний вигляд:

$$F = \sum_{j=1}^{n} c_j x_j \to \max, \tag{2.1}$$

$$\sum_{j=1}^{n} a_j x_j \le b \tag{2.2}$$

де x_j – кількість предметів типу j;

c_j – прибуток від перевезення одного предмету типу j;

a_j – вага предмету типу j;

b – вантажопідйомність транспорту;

F – прибуток від перевезення обраних предметів типу j у кількості x_j відповідно без перевищення вантажопідйомності.

Як приклад розглянемо задачу про завантаження транспорту з наступними вхідними даними:

$$b = 10;\ a_1 = 5;\ a_2 = 2;\ a_3 = 3;\ c_1 = 100;\ c_2 = 40;\ c_3 = 80.$$

2.2 Методи розв’язання задачі

2.2.1 Метод повного перебору

Існує очевидний і досить універсальний метод розв’язування оптимізаційних задач, який можна використовувати, коли множина можливих рішень M обмежена. Це комплексний метод ранжирування, який ранжує всі можливі варіанти. Метод забезпечує гарантований розв’язок, коли множина M обмежена (типова ситуація дискретного програмування) й є ефективним алгоритмом для генерації будь-якого елемента з M і обчислення цільової функції цього елемента.

Тому ми маємо типовий загальноінтелектуальний підхід. Коли інтелектуальна система потрапляє в нову ситуацію та намагається спланувати майбутні дії, вона може спробувати звести задачу планування навмисних дій до задачі оптимізації (для цього, звичайно, достатньо визначити множину можливих розв’язків M і цільову функцію $f(x)$). У разі успіху ретельний розгляд варіантів у більшості випадків призведе до ідеального рішення.

Однак повне перерахування має очевидний недолік: у більшості практичних ситуацій кількість варіантів сортування дуже велика, і цей метод не представляється можливим для реалізації в розумний проміжок часу. Тому були розроблені методи та алгоритми для обмеження таких пошуків.

У той же час, не завжди потрібно шукати ідеальне рішення. Часто буває досить обмежитися неоптимальним розв’язком (близьким до ідеального) або просто прийнятним. У таких випадках достатньо розробити правило, яке усуває необхідність повного пошуку та забезпечує прийнятний результат для більшості документів.

Однак жодна інтелектуальна проблема не призводить до очевидного зменшення проблеми оптимізації в будь-якому випадку, оскільки часто неможливо явно записати цільову функцію або обмеження.

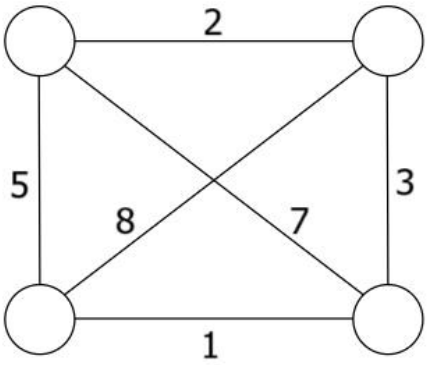
Метод повного перебору є просто методом грубої сили для комбінаторних задач. Він пропонує генерувати кожен елемент області, вибирати ті, які

задовольняють усі обмеження, а потім знаходити бажаний елемент (наприклад, той, який оптимізує дану цільову функцію). Зауважте, що хоча ідея методу повного перебору досить проста, для її реалізації зазвичай потрібен алгоритм, який генерує комбінаторні об’єкти [13].

Проілюструємо метод повного перебору, застосувавши його до трьох важливих проблем: задача комівояжера, задача про рюкзак та задача призначення.

Проблема комівояжера. Задача комівояжера (TSP) цікавить дослідників протягом 150 років через її начебто просте формулювання, важливі застосування та цікаві зв’язки з іншими комбінаторними проблемами. Простіше кажучи, проблема полягає в тому, щоб знайти найкоротший маршрут через заданий набір із n міст, який проходить через кожне місто рівно один раз, перш ніж повернеться до міста, з якого він почався. Проблему можна легко змоделювати за допомогою зваженого графа, де вершини графа представляють міста, а ваги ребер визначають відстані. Тоді задачу можна сформулювати як задачу знаходження найкоротшого контуру гамільтонового графіка. (Шум Гамільтона визначається як цикл, який проходить усі вершини графа точно один раз. Його назва походить від імені ірландського математика сера Вільяма Роуена Гамільтона (1805-1865), який цікавився цими циклами як застосуванням своїх алгебраїчних відкриттів.)

Легко побачити, що гамільтонів контур також можна визначити як послідовність $n+1$ суміжних вершин v_{i0}, v_{i1}, ..., v_{in-1}, де перша вершина послідовності збігається з останньою, а всі решта $n+1$ вершин різні. Крім того, ми можемо припустити без втрати загальності, що всі схеми починаються і закінчуються в певній вершині. Отже, ми можемо отримати всі схеми, згенерувавши всі $n-1$ перестановок проміжних міст, обчислити довжину схеми та знайти найкоротшу. На рисунку 2.1 зображено невеликий приклад задачі та її розв’язання цим методом.



Шлях	Довжина	
$a \dashrightarrow b \dashrightarrow c \dashrightarrow d \dashrightarrow a$	$l = 2 + 8 + 1 + 7 = 18$	
$a \dashrightarrow b \dashrightarrow d \dashrightarrow c \dashrightarrow a$	$l = 2 + 3 + 1 + 5 = 11$	Оптимальне
$a \dashrightarrow c \dashrightarrow b \dashrightarrow d \dashrightarrow a$	$l = 5 + 8 + 3 + 7 = 23$	
$a \dashrightarrow c \dashrightarrow d \dashrightarrow b \dashrightarrow a$	$l = 5 + 1 + 3 + 2 = 11$	Оптимальне
$a \dashrightarrow d \dashrightarrow b \dashrightarrow c \dashrightarrow a$	$l = 7 + 3 + 8 + 5 = 23$	
$a \dashrightarrow d \dashrightarrow c \dashrightarrow b \dashrightarrow a$	$l = 7 + 1 + 8 + 2 = 18$	

Рисунок 2.1 – Приклад задачі комівояжера

Дивлячись на цей рисунок, ми бачимо три пари маршрутів, які відрізняються лише напрямком. Таким чином, ми могли б вдвічі зменшити кількість перестановок вершин. Ми могли б, наприклад, вибрати будь-які дві проміжні вершини, скажімо b й c, а потім розглянути лише ті перестановки, у яких b передує c. Однак це може не значно покращити продуктивність. Загальна кількість необхідних перестановок завжди дорівнює $(n-1)!$, що робить підхід вичерпного пошуку непрактичним для всіх, крім дуже малих значень n. З іншого боку, якщо ви завжди бачите склянку наполовину повною, ви можете заперечити, що не варто чхати, щоб скоротити свою роботу навпіл, навіть якщо ви вирішуєте невелику проблему, особливо з рукою. Слід також зазначити, що якби ми не обмежували наше дослідження шаблонами, починаючи з однієї вершини, кількість перестановок була б ще більшою, у n разів.

Задача про завантаження рюкзака. Це ще одна відома задача алгоритму. Дано n елементів з відомими вагами w_1, w_2, ..., w_n та значення v_1, v_2, ..., v_n, та рюкзак місткістю W, потрібно знайти найціннішу підмножину предметів у рюкзаку. На рисунку 2.2 показаний невеликий приклад проблеми з рюкзаком.

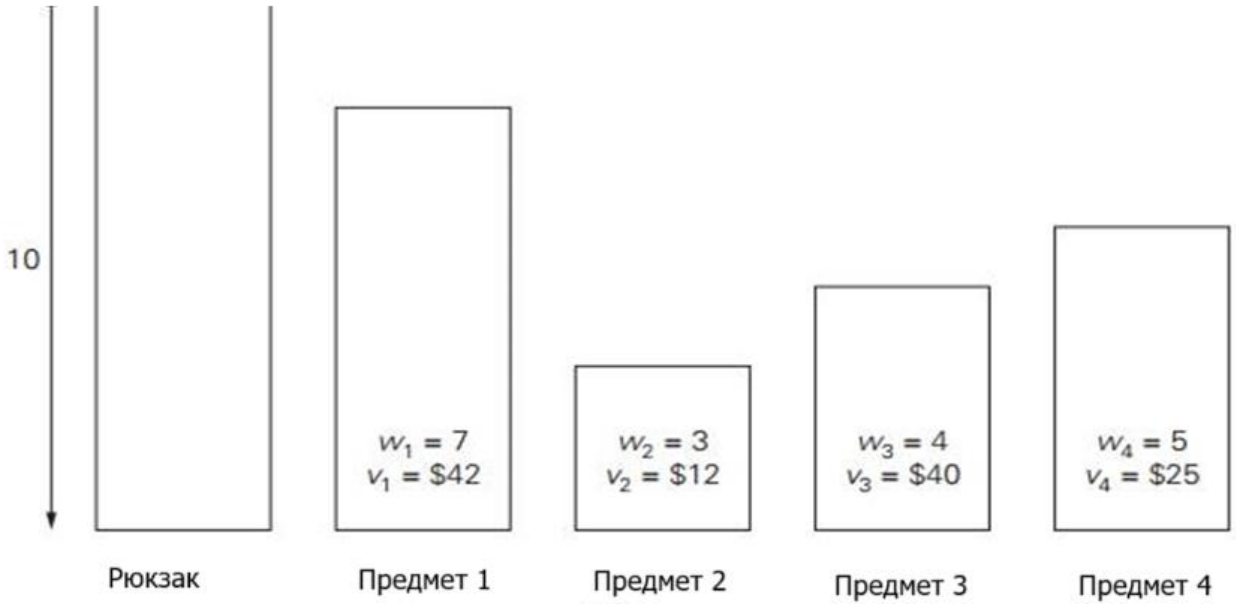


Рисунок 2.2 – Приклад задачі про рюкзак

Набір	Загальна маса	Загальна вартість
∅	0	$ 0
{1}	7	$42
{2}	3	$12
{3}	4	$40
{4}	5	$25
{1, 2}	10	$54
{1, 3}	11	нездійсненно
{1, 4}	12	нездійсненно
{2, 3}	7	$52
{2, 4}	8	$37
{3, 4}	**9**	**$65**
{1, 2, 3}	14	нездійсненно
{1, 2, 4}	15	нездійсненно
{1, 3, 4}	16	нездійсненно
{2, 3, 4}	12	нездійсненно
{1, 2, 3, 4}	19	нездійсненно

Рисунок 2.3 – Розв’язок задачі про рюкзак

Підхід методу повного перебору до цієї проблеми генерує всі підмножини набору з n даних предметів, обчислює загальну вагу кожної підмножини, щоб

ідентифікувати життєздатні підмножини (тобто ті, загальна вага яких не перевищує місткості рюкзака), та визначає найцінніші підмножини серед них. Наприклад, розв'язок прикладу на рисунку 2.2 показано на рисунку 2.3. Оскільки кількість підмножин набору з n елементів дорівнює $2n$, метод повного перебору створює $(2n)$-алгоритм, незалежно від того, наскільки ефективно генеруються окремі підмножини [6].

Таким чином, і в задачі комівояжера, і в задачі рюкзака, що розглянуті вище, метод повного перебору призводить до надзвичайно неефективних алгоритмів для кожного входу. Насправді ці дві задачі є найвідомішими прикладами так званих NP-складних задач. Немає відомого поліноміального алгоритму для будь-якої NP-складної задачі. Крім того, більшість програмістів вважають, що таких алгоритмів не існує, хоча ця дуже важлива гіпотеза ніколи не була доведена. Досконаліші підходи – відстеження назад і розгалуження – дозволяють нам вирішити деякі, але не всі, випадки цих та подібних проблем за менш ніж експоненціальний час.

Задача про призначення. У нашому третьому прикладі перерахування є n людей, яким необхідно призначити n робочих місць, по одній особі на робоче місце (рис. 2.4**)**. (Це означає, що кожній особі призначається рівно одна робота та кожна робота призначається точно одній особі.) Вартість, яка буде стягнута, коли i-ту особу призначають на j-ту роботу, є відомою величиною $C[i, j]$ для кожної пари $i, j = \overline{1; n}$. Проблема полягає в тому, щоб знайти діяльність з мінімальними загальними витратами.

Нижче наведено невеликий приклад цієї проблеми із записами таблиці, що представляють витрати на призначення $C[i, j]$:

	Робота 1	Робота 2	Робота 3	Робота 4
Особа 1	9	2	7	8
Особа 2	6	4	3	7
Особа 3	5	8	1	8
Особа 4	7	6	9	4

Рисунок 2.4 – Приклад задачі про призначення

Легко бачити, що приклад задачі призначення повністю задано матрицею вартості C. Для цієї матриці проблема полягає в тому, щоб вибрати елемент у кожному рядку матриці так, щоб усі вибрані елементи були в різних стовпцях, а загальна кількість вибраних елементів була мінімальною. Зауважимо, що жодні очевидні стратегії вирішення тут не працюють. Наприклад, ми не можемо вибрати найменший елемент у кожному рядку, оскільки менші елементи можуть бути в одному стовпці. Насправді, найменший елемент усієї матриці не обов'язково є компонентом оптимального розв'язку. Таким чином, рішення провести вичерпний пошук може виявитися неминучим злом.

Можливі розв'язки задачі призначення можна описати у вигляді кортежів $j_1, ..., j_n$, де i-та компонента, $i = 1, 2, 3, ..., n$, вказує на стовпець вибраного пункту в i-му рядку (тобто номер роботи, призначений i-й особі). Наприклад, у вищезазначеній матриці витрат цифри 2, 3, 4, 1 вказують на призначення особи 1 на посаду 2, людину 2 на позицію 3, особу 3 на позицію 4 і особу 4 на позицію 1. Вимоги до призначення означають, що існує взаємна відповідність між можливими призначеннями та перестановками перших n цілих чисел.

Таким чином, підхід повного перерахування до проблеми призначення вимагав би створення всіх перестановок цілих чисел $1, 2, 3, ..., n$, обчислення загальної вартості кожної діяльності шляхом додавання відповідних елементів матриці витрат і, нарешті, вибору того, що має найменшу суму. Перші ітерації застосування цього алгоритму до попереднього випадку представлені на рисунку 2.5.

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

<1, 2, 3, 4> cost = 9 + 4 + 1 + 4 = 18
<1, 2, 4, 3> cost = 9 + 4 + 8 + 9 = 30
<1, 3, 2, 4> cost = 9 + 3 + 8 + 4 = 24
<1, 3, 4, 2> cost = 9 + 3 + 8 + 6 = 26
<1, 4, 2, 3> cost = 9 + 7 + 8 + 9 = 33
<1, 4, 3, 2> cost = 9 + 7 + 1 + 6 = 23

Рисунок 2.5 – Перші ітерації методу повного перебору для задачі про призначення

Оскільки кількість перестановок, які слід розглянути в загальному випадку проблеми призначення, дорівнює $n!$, метод повного перебору не є практичним для всіх, крім дуже малих випадків проблеми.

2.2.2 Жадібний алгоритм

Жадібний алгоритм – це інтуїтивно зрозумілий і ефективний спосіб розв'язання проблем оптимізації. І хоча його реалізація приваблива своїми результатами, вона не завжди є оптимальною. Важливо точно знати, коли використовувати жадібний підхід, а коли його уникати.

Жадібні алгоритми – це ціле сімейство алгоритмів (іноді їх називають жадібним підходом або жадібним програмуванням), тому не існує конкретного жадібного алгоритму, який можна було б закодувати. Проте всі жадібні алгоритми побудовані на одному принципі: вибір оптимального розв'язку на кожному кроці, незалежно від кроків, які були зроблені раніше або будуть зроблені. Іншими словами, жадібний алгоритм робить локально оптимальний вибір у надії, що він призведе до глобально оптимального рішення.

Переваги та недоліки. Жадібний підхід легко зрозуміти та закодувати. На кожному етапі алгоритму ми можемо ігнорувати попередні та наступні кроки та думати лише про оптимальне рішення на цьому етапі. Жадібний підхід не передбачає скасування вибору (відтворення попередніх кроків) і не передбачає нічого на майбутнє.

Швидкість виконання програми при жадібному підході легко передбачити, оскільки складність алгоритму очевидна. Здебільшого він є лінійним, тобто час виконання програми лінійно залежить від кількості вхідних даних. Це не завжди стосується інших алгоритмічних підходів, таких як «Розділяй і володарюй».

Але такий підхід має істотний недолік. У більшості випадків жадібний алгоритм не працює належним чином. Потрібно добре розуміти, коли його можна використовувати, а коли ні. І навіть якщо в деяких випадках жадібний алгоритм забезпечує оптимальний розв'язок, важко довести, що цей підхід працюватиме у всіх інших можливих випадках.

У прикладі монет жадібний алгоритм добре працює для монет номіналом 1, 2, 5, але не працює для номіналів 1, 2, 5, 6. Потрібно зауважити, що всі монетні системи розроблені таким чином. щоб жадібний алгоритм працював для них правильно. Оскільки це швидко та просто, люди легко знаходять потрібну суму для розрахунку в супермаркеті.

Принцип застосування. Для розуміння використання жадібного підходу існує евристичний принцип. Якщо наступні дві властивості вірні, проблему можна вирішити за допомогою жадібного алгоритму:

- жадібний принцип відбору. Набір оптимальних варіантів на кожному етапі в кінцевому рахунку призводить до оптимального рішення;
- оптимальна структура субстрату. Проблема має оптимальну підструктуру, якщо оптимальне рішення всієї проблеми містить оптимальне рішення кожної підпроблеми. Іншими словами, після завершення певного кроку алгоритм продовжує розв’язувати проблему, для чого також працює жадібний підхід.

Обмеження жадібних алгоритмів

- локальна оптимальність: жадібні алгоритми намагаються знайти локально оптимальне рішення на кожному кроці, що не завжди призводить до глобально оптимального рішення;
- недостатньо глобальної інформації: жадібні алгоритми приймають рішення на основі локальних виборів, не беручи до уваги всю глобальну інформацію.

Приклади жадібних алгоритмів

- жадібний алгоритм для проблеми мінімального остовного дерева: прикладом використання жадібних алгоритмів є проблема MST. Це завдання передбачає пошук найкоротшого шляху через усі вершини графа без створення циклів. Жадібний алгоритм MST починається з однієї вершини та крок за кроком додає найкоротше доступне ребро, яке не утворює цикл. Результат – найменше остовне дерево;
- жадібний алгоритм для проблеми рюкзака: проблема рюкзака виникає, коли вибрано набір предметів з обмеженим об’ємом. Алгоритм

проблеми з жадібним рюкзаком вибирає продукти з найвищою ціною за одиницю ваги та додає їх у рюкзак, доки не буде досягнуто обмеження обсягу;

– жадібний алгоритм для проблеми найкоротшого шляху: у задачі найкоротшого шляху жадібний алгоритм вибирає вершину, найближчу до поточної точки, і додає її до шляху. Він продовжує цей процес, доки не буде досягнуто кінцевої точки або доки більше не буде доступних точок.

Порівняння з іншими типами алгоритмів

– жадібні алгоритми та динамічне програмування: динамічне програмування вирішує проблеми, розбиваючи їх на менші підпроблеми та зберігаючи результати цих підзадач для подальшого використання. Порівняно з жадібними алгоритмами, динамічне програмування може знайти глобально оптимальне рішення, але може бути складнішим у реалізації та вимагати більше обчислювальних ресурсів;
– жадібні алгоритми проти методів грубої сили (методу повного перебору): метод повного перебору вирішує проблему шляхом перегляду всіх можливих комбінацій рішень. У порівнянні з жадібними алгоритмами, ітераційний алгоритм знаходить глобально оптимальне рішення за допомогою Takuu, але він вимагає високої обчислювальної потужності, особливо для великих проблем.

Використання жадібних алгоритмів в реальному світі

– жадібні алгоритми в обчислювальній техніці: Жадібні алгоритми зазвичай використовуються в обчислювальній техніці, наприклад в оптимізації мережі, плануванні ресурсів, управлінні даними та в інших сферах;
– жадібні алгоритми в маркетингу: у маркетингу жадібні алгоритми можна використовувати для оптимізації рекламних кампаній, визначення стратегій ціноутворення тощо. Наприклад, визначаючи

оптимальне розміщення реклами на веб-сайті, жадібний алгоритм може вибрати рекламу, яка приносить найбільший дохід.

Жадібні алгоритми є ефективним способом розв’язання проблеми оптимізації шляхом вибору найкращого доступного варіанту на кожному кроці. Він має свої переваги та недоліки, але може бути використаний у кількох сферах, включаючи IT та маркетинг. Використовуючи жадібні алгоритми, можна досягти швидких і прийнятних результатів при вирішенні задач оптимізації [13].

2.2.3 Метод динамічного програмування: ключові аспекти та застосування

Якщо ви коли-небудь стикалися з проблемами, які потребують визначення найкращого рішення на основі обмежень, ви, ймовірно, знаєте, що для цього існує метод динамічного програмування. Це досить потужний пристрій, який можна використовувати для вирішення складних завдань, починаючи від знаходження найбільшої спільної підпослідовності та закінчуючи визначенням найвигіднішої комбінації продуктів у рюкзаку. Якщо ви хочете навчитися використовувати цей метод, вам потрібно зрозуміти, як працює динамічний програміст.

Термін «динамічне програмування» був придуманий і названий у 1940 році Річардом Беллманом, а його визначення було змінено та розширено в 1953 році. Беллману довелося витратити багато часу на вибір назви, тому що його начальнику не подобався термін «математика». Тому автор визначення вибрав слово «програмування» замість «дизайн» і слово «динамічний», щоб уникнути принизливого та образливого тлумачення з боку керівника. Так народилася назва «динамічне програмування» [14].

Динамічне програмування (ДП) є одним з найважливіших інструментів для оптимізації та вирішення проблем в інформатиці, економіці, біології та інших галузях. Простіше кажучи, динамічне програмування – це метод розв’язування задач, який передбачає розбиття складної проблеми на кілька менших проблем;

Важливо враховувати наступні моменти:

а) щоб ефективно використовувати цей підхід, потрібно пам’ятати про вирішення підпроблем;

б) підзадачі мають загальну структуру, яка дозволяє використовувати уніфікований спосіб їх вирішення, а не розв’язувати їх окремо за допомогою різних алгоритмів.

За допомогою ДП проблеми оптимізації вирішуються ефективно, наприклад, якщо ви хочете знайти максимальне або мінімальне значення функції. Також ДП активно використовується при плануванні завдань, де необхідно визначити оптимальну послідовність дій.

Одним із основних понять є ідеальна підструктура. У методі ДП ми вирішуємо проблему, розділяючи її на менші задачі. Оптимальна підструктура означає, що ми можемо отримати найкраще рішення, якщо знаємо оптимальні рішення для кожної підпроблеми.

Ще одна важлива – перекриття підзавдань. Ми можемо зіткнутися з ситуацією, коли різні підзадачі мають спільні частини. У цьому випадку ми говоримо, що у нас є підзавдання, що збігаються. Щоб уникнути розв’язання однієї й тієї ж задачі кілька разів, ми зберігаємо результати підзадач у пам’яті та повторно використовуємо їх під час розв’язання більших задач. Це значно прискорить процес вирішення.

ДП – це методологія вирішення проблем, яка є не просто формулою чи алгоритмом, а міркуванням про те, як вирішити проблему. Цей підхід передбачає поділ проблеми на менші, простіші підпроблеми (з меншими вхідними даними, такими як менша кількість, менша група або менша кількість параметрів для визначення).

Рішення невеликих підпроблем можуть вирішити більшу вихідну проблему. Прикладом може бути обчислення чисел Фібоначчі.

При цьому важливо ефективно використовувати розв’язки підзадач, наприклад, шляхом запам’ятовування, і використовувати єдиний метод розв’язування всіх розв’язків підзадач, якщо вони мають загальну структуру.

Алгоритм методу ДП складається з кількох кроків:

а) визначити структуру оптимізаційної задачі. Необхідно визначити, які параметри задачі є змінними, які константами і які обмеження має змінна;

б) сформулюйте рекурсивну формулу. Необхідно виразити вирішення проблеми через вирішення більш дрібних підзадач. Рекурсивна формула повинна бути дійсною і мати властивість оптимальної підструктури;

в) створення таблиці для зберігання результатів виконання підзадач. Необхідно створити таблицю, в кожній клітинці якої зберігається оптимальний розв’язок відповідної підзадачі;

г) заповніть таблицю. Заповніть таблицю, починаючи з найменших підзадач і поступово переходячи до більших. При заповненні таблиці використовується рекурсивна формула;

д) отримайте рішення вихідної проблеми. Рішення вихідної задачі знаходиться в останній клітинці таблиці.

Плюси і мінуси. Як і будь-який інший алгоритм, цей метод має переваги та недоліки.

Переваги:

– точність: на відміну від алгоритмів, які приймають локально оптимальні рішення, ДП може гарантувати глобально оптимальне рішення;
– гнучкість: може бути застосована до широкого кола завдань (пошук найкоротшого шляху в графі, оптимальний дизайн задачі та багато інших);
– ефективність: ДП може бути дуже ефективним для вирішення проблем, які можна розділити на підпроблеми, і де оптимальне рішення для кожної підпроблеми можна обчислити лише один раз.

Недоліки:

– висока складність: ДП може мати високу обчислювальну складність, особливо якщо вихідна проблема дуже складна і може бути розділена на кілька підпроблем;

- високі вимоги до пам’яті: значення всіх підзадач повинні зберігатися в пам’яті, що може призвести до великого споживання пам'яті.
- необхідність певної структури проблеми: ДП може бути неефективним або невідповідним, якщо вихідна проблема не розділена на підпроблеми або якщо їх вирішення не можна ефективно поєднати для отримання основного рішення.

У майбутньому метод ДП буде продовжувати використовуватися в різних сферах, таких як фінанси, виробництво, транспорт і багато інших, для вирішення складних задач оптимізації. Однак дослідження необхідно продовжувати для розробки нових, більш ефективних і універсальних алгоритмів для вирішення різних проблем [14].

2.3 Розв’язок детермінованої задачі про завантаження транспорту

Існує кілька відомих алгоритмів [7] для розв’язання задачі завантаження транспорту за наведеною вище умовою задачі.

Знайдемо розв’язок задачі методом перебору даних (табл. 2.1). Перші три стовпці x_j відповідають кількості елементів типу j-го завантажених у транспорт. Оскільки вага x_3 одного елемента третього типу дорівнює 3, то максимальна кількість одиниць цього типу, які можна завантажити, дорівнює $\left[\frac{10}{3}\right]=3$. Це означає, що для x_3 можливі значення дорівнюють 0, 1, 2, 3. Аналогічним способом розрахуємо значення для об’єктів іншого типу x_2 (оскільки вага одного об’єкта іншого типу x_2 дорівнює 2, то максимальна кількість пар цього типу $\left[\frac{10}{2}\right]=5$, тобто набуватимуть наступних значень 0, 1, 2, 3, 4, 5); аналогічно і для першого типу x_1 (оскільки вага x_1 становить 5, то максимальна кількість одиниць цього типу $\left[\frac{10}{5}\right]=2$, тобто значення будуть дорівнювати 0, 1, 2).

У четвертому стовпчику розраховуємо місткість транспорту для кожного елемента x_j. У п’ятому стовпчику розраховуємо дохід від перевезення предметів, маса яких не перевищує місткості транспорту.

Таблиця 2.1 – Ручний розрахунок

x_1	x_2	x_3	$5x_1+2x_2+3x_3 \le 10$	$100x_1+40x_2+80x_3 \to \max$
0	0	0	0	0
0	0	1	3	80
0	0	2	6	160
0	0	3	9	240
0	1	0	2	40
0	1	1	5	120
0	1	2	8	200
0	1	3	11	280
0	2	0	4	80
0	2	1	7	160
0	2	2	10	240
0	2	3	13	320
0	3	0	6	120
0	3	1	9	200
0	3	2	12	280
0	3	3	15	360
0	4	0	8	160
0	4	1	11	240
0	4	2	14	320
0	4	3	17	400
0	5	0	10	200
0	5	1	13	280
0	5	2	16	360
0	5	3	19	440
1	0	0	5	100
1	0	1	8	180
1	0	2	11	260
1	0	3	14	340
1	1	0	7	140
1	1	1	10	220
1	1	2	13	300
1	1	3	16	380
1	2	0	9	180
1	2	1	12	260
1	2	2	15	340

Продовження таблиці 2.1

1	2	3	18	420
1	3	0	11	220
1	3	1	14	300
1	3	2	17	380
1	3	3	20	460
1	4	0	13	260
1	4	1	16	340
1	4	2	19	420
1	4	3	22	500
1	5	0	15	300
1	5	1	18	380
1	5	2	21	460
1	5	3	24	540
2	0	0	10	200
2	0	1	13	280
2	0	2	16	360
2	0	3	19	440
2	1	0	12	240
2	1	1	15	320
2	1	2	18	400
2	1	3	21	480
2	2	0	14	280
2	2	1	17	360
2	2	2	20	440
2	2	3	23	520
2	3	0	16	320
2	3	1	19	400
2	3	2	22	480
2	3	3	25	560
2	4	0	18	360
2	4	1	21	440
2	4	2	24	520
2	4	3	27	600
2	5	0	20	400
2	5	1	23	480
2	5	2	26	560
2	5	3	29	640

Визначимо набори значень, які не перевищують навантаження транспорту та дають максимальний дохід від транспортування. Найкращі набори для

завантаження транспорту, що задовольняють обмеженням – це два набори: $(0;0;3)$, $(0;2;2)$ та максимальний прибуток цих наборів, що становить 240.

Знайдемо розв'язок задачі методом ДП (табл. 2.2-2.4). Розглянемо спочатку невідомі коефіцієнти x_j у формулі (2.1) (обмеження у математичній постановці задачі). Знайдемо цілу частину a_j за підйомною силою транспорту (в нашому випадку підйомна сила $b=10$). Оскільки вага x_3 одного елемента третього типу дорівнює 3, то максимальна кількість одиниць цього типу, які можна завантажити, становить $\left[\frac{10}{3}\right]=3$. Це означає, що можливі значення для x_3 будуть 0, 1, 2, 3. У таблиці 2.2 y_j – здатність транспорту перевозити вантаж (стовпець 1), маса третього об'єкта 1 тонна ($a_3=1$). Можна взяти від нуля до п'яти одиниць вантажу включно. Отже, ми перебираємо всі можливі товари x_j. Якщо ми не беремо об'єкти третього типу, то прибуток у всіх випадках дорівнює нулю одиниць (стовпчик 2). Якщо ми беремо один об'єкт третього типу, то зрозуміло, що він не забере нуль одиниць, а забере одну одиницю навантаження транспорту, але все одно заповнюємо всю таблицю. В таблицю ми заносимо дохід від перевезення предмету цього третього виду, вартість від передачі одного предмета становить 80 одиниць: $c_3=80$ (стовпчик 3). Далі беремо два елемента третього типу, оскільки ми візьмемо відразу дві тони вантажопідйомності, то в перші два рядки будуть порожні. А в наступних рядках (стовпчик 4) матимемо дохід 160 одиниць і так далі.

У стовпчику 6 вибираємо максимальний прибуток для будь-якої кількості елементів третього типу. У сьомому стовпчику записуємо, якому x_j відповідає $f_3(y_3)$ (див. табл. 2.2).

Таблиця 2.2 – Предмети третього типу для завантаження транспорту

1	2	3	4	5	6	7
y_3	$x_0=0$	$x_1=1$	$x_2=2$	$x_3=3$	$f_3(y_3)$	x_3^*

Продовження таблиці 2.2

0	0	–	–	–	0	0
1	0	–	–	–	0	1
2	0	–	–	–	0	2
3	0	80	–	–	80	3
4	0	80	–	–	80	4
5	0	80	–	–	80	5
6	0	80	160	–	160	6
7	0	80	160	–	160	7
8	0	80	160	–	160	8
9	0	80	160	240	240	9
10	0	80	160	240	240	10

Для наступної таблиці ми маємо характерну величину x_j, оскільки ціла частина другого типу дорівнює $\left[\frac{10}{2}\right]=5$. Це означає, що x_j зміниться від 0 до 5. Величина y_j не зміниться для нас, тому що ми можемо взяти від нуля до п'яти тон для кожного елемента. Потім будемо дотримуватися того ж правила, що й для предметів третього типу. Якщо ми не беремо предмети другого типу, то вантажопідйомність не зміниться (стовпчик 2). Якщо взяти один другого типу, але його маса 2 тони ($a_2=1$), то один предмет другого типу відразу займе дві тони вантажопідйомності транспорту, тобто в перших двох рядках таблиці 2.3 маємо порожні поля, але дохід від перевезення предметів другого виду становить 40 одиниць ($c_2=40$), тоді отримуємо прибуток від перевезення одного предмета другого виду 40 одиниць. Як і в стовпці 4, якщо ми візьмемо два елементи другого типу, ми відразу почнемо опрацьовувати можливість заповнення таблиці з четвертого рядка, але потрібно врахувати, який прибуток ми вже маємо від цього, тому додаємо прибуток від перевезення предметів третього типу (додаємо графу 6 таблиці 2.2 доходів від перевезення предметів другого виду, починаючи з першого рядка). Стовбець 9 таблиці 2.3 заповнюємо відповідно як стовбець 7 таблиці 2.2, тобто якому x_j відповідає $f_2(y_2)$.

Таблиця 2.3 – Предмети другого та третього типів для завантаження транспорту

1	2	3	4	5	6	7	8	9
y_2	$x_0 = 0$	$x_1 = 1$	$x_2 = 2$	$x_3 = 3$	$x_4 = 4$	$x_5 = 5$	$f_2(y_2)$	x_2^*
0	0+0	–	–	–	–	–	0	0
1	0+0	–	–	–	–	–	0	0
2	0+0	40+0	–	–	–	–	40	1
3	0+80	40+0	–	–	–	–	80	0
4	0+80	40+0	80+0	–	–	–	120	0,2
5	0+80	40+80	80+0	–	–	–	120	1
6	0+160	40+80	80+0	120+0	–	–	160	0
7	0+160	40+80	80+80	120+0	–	–	160	0,2
8	0+160	40+160	80+80	120+0	–	–	200	1
9	0+240	40+160	80+80	120+80	160+0	–	240	0
10	0+240	40+160	80+160	120+80	160+0	200+0	240	0,2

Відповідно таблицю 2.4 ми заповнюємо даними, але з урахуванням ваги $a_1 = 5$ та ціни $c_1 = 100$ для отримання прибутку від перевезення предметів першого типу, але з урахуванням прибутку від перевезення предметів другого та третього типів разом (табл. 2.3).

Таблиця 2.4 – Предмети перших-третіх типів для завантаження літака

1	2	3	4	5	6
y_1	$x_0 = 0$	$x_1 = 1$	$x_2 = 2$	$f_1(y_1)$	x_1^*
0	0+0	–	–	0	0
1	0+0	–	–	0	0
2	0+40	–	–	40	0
3	0+80	–	–	80	0
4	0+80	–	–	80	0
5	0+120	100+0	–	120	0
6	0+160	100+0	–	160	0
7	0+160	100+40	–	160	0

Продовження таблиці 2.4

8	0+200	100+80	–	200	0
9	0+240	100+80	–	240	0
10	0+240	100+120	200+0	240	0

Знайдемо тепер оптимальний план для задачі. Спочатку вибираємо із таблиці 2.4 максимальне значення $f_1(y_1)$, яке дорівнює 240. В нашому випадку максимальне значення відповідає одному значенню x_j ($x_0 = 0$). Оптимальний розв'язок матиме вид:

$$x_1^* = 0.$$

Далі виписуємо обмеження (2.2), але замінюємо знак «менше» на «дорівнює», при цьому підставляємо значення x_1^*:

$$5x_1 + 2x_2 + 3x_3 = 10,$$
$$2x_2 + 3x_3 = 10 - 5x_1,$$
$$2x_2 + 3x_3 = 10.$$

Переходимо до таблиці 2.3 та вибираємо x_j, що відповідає значенню $f_2(y_2)$, яке дорівнює 10. Оскільки максимальний дохід дають нам цілих два варіанти x_j: $x_0 = 0$ та $x_2 = 2$, то нам треба перебрати цілих два варіанта наборів.

Вибираємо $x_2^* = 0$ і знову підставляємо в обмеження (2.2), але з урахуванням x_1^* і знаходимо x_3^*:

$$5x_1 + 2x_2 + 3x_3 = 10 \Rightarrow$$
$$3x_3 = 10 - 5x_1 - 2x_2,$$
$$3x_3 = 10 \Rightarrow x_3^* = 3.$$

Сформуємо оптимальний набір із отриманих результатів:

$$x_1^* = x_2^* = 0,\ x_3^* = 3.$$

Отже, першим оптимальним набором буде набір $(0; 0; 3)$.

Аналогічним способом отримаємо другий оптимальний набір:

$$x_1^* = 0,\ x_2^* = 2,$$

$$5x_1 + 2x_2 + 3x_3 = 10 \Rightarrow$$

$$3x_3 = 10 - 5x_1 - 2x_2$$

$$3x_3 = 6 \Rightarrow x_3^* = 2 \Rightarrow (0; 2; 2).$$

Отже, в результаті було отримано два оптимальних набори $(0; 0; 3)$ та $(0; 2; 2)$, які співпадають з відповідями, отриманими методом перебору даних.

2.4 Програмна реалізація задачі про завантаження транспорту в середовищі MATLAB

Для написання програмного коду використовувався пакет MATLAB [15]. Було написано два програмних коди: програмний код методу перебору даних (додаток А) і програмний код динамічного програмування (додаток Б).

Вхідними даними для програми методу класифікації даних є матриця C – матриця транспортних витрат. У програмі ми проходимо всі можливі розв'язки, для яких розраховуємо транспортний дохід. Далі перевіряємо, чи не перевищують розв'язки значення вантажопідйомності транспорту, чи задовольняють вони наше обмеження в математичній постановці задачі та дають

максимальний транспортний прибуток, потім виводимо цей набір елементів на екран. У результаті виконання програми для методу перебору даних ми отримаємо наступний результат (див. рис. 2.6). На виході програми отримуємо максимальне значення цільової функції та два оптимальних набори елементів.

```
Price:
c =
   100    40    80

f_optimal =
   240
x_optimal =
     0     0     3
     0     2     2

answer:
x_optimal =
     0     0     3
     0     2     2
f_optimal =
   240
Elapsed time is 0.326958 seconds.
```

Рисунок 2.6 – Результат роботи програми методу перебору даних

На вході визначаємо матрицю C (матриця транспортних витрат). В результаті ми отримуємо різницю в методі класифікації даних. Ось ми отримуємо перший із оптимальних наборів. Результатом виконання програми методом динамічного програмування є такий результат (рис. 2.7).

```
Price =
   100    80    40

Answer:

     y     F     X              F_OPT
     0     0     0     0  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     1    40     1     0    40  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     2    80     2     0    40    80  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     3   120     3     0    40    80   120  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     4   160     4     0    40    80   120   160  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     5   200     5     0    40    80   120   160   200  -Inf  -Inf  -Inf  -Inf  -Inf
     6   240     6     0    40    80   120   160   200   240  -Inf  -Inf  -Inf  -Inf
     7   280     7     0    40    80   120   160   200   240   280  -Inf  -Inf  -Inf
     8   320     8     0    40    80   120   160   200   240   280   320  -Inf  -Inf
     9   360     9     0    40    80   120   160   200   240   280   320   360  -Inf
    10   400    10     0    40    80   120   160   200   240   280   320   360   400

     y     F     X              F_OPT
     0     0     0     0  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     1    40     0    40  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     2    80     0    80    80  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     3   120     0   120   120  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     4   160     0   160   160   160  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     5   200     0   200   200   200  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     6   240     0   240   240   240   240  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     7   280     0   280   280   280   280  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     8   320     0   320   320   320   320   320  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
     9   360     0   360   360   360   360   360  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
    10   400     0   400   400   400   400   400   400  -Inf  -Inf  -Inf  -Inf  -Inf

   y     F     X              F_OPT
   0     0     0     0  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   1    40     0    40  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   2    80     0    80  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   3   120     0   120  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   4   160     0   160  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   5   200     0   200   100  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   6   240     0   240   140  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   7   280     0   280   180  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   8   320     0   320   220  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
   9   360     0   360   260  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf
  10   400     0   400   300   200  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf

x_optimal =
     0     0     3
f_optimal =
   240
Elapsed time is 0.016092 seconds.
```

Рисунок 2.7 – Результат програми методу динамічного програмування

Порівнюючи результати розв’язування детермінованої задачі ручним розрахунком і розв’язування її за допомогою програм, бачимо, що відповіді збігаються.

2.5 Програмна реалізація задачі про завантаження транспорту за допомогою мови JAVA

Нижче наведений результат програмного коду на мові Java (додаток В), який реалізує розв’язання задачі про завантаження транспорту методом динамічного програмування (рис. 2.8).

```
C:\Users\Administrator\.jdks\openjdk-20.0.1\bin\java.exe
Dynamic_metod
Оптимальний набір
0 0 3
Максимальний прибуток
240
Час виконання: 0.037437 ms.

Process finished with exit code 0
```

Рисунок 2.8 – Результат програми методу динамічного програмування мовою JAVA

Нижче наведений результат програмного коду на мові Java (додаток Г), який реалізує розв’язання задачі про завантаження транспорту методом повного перебору (рис. 2.9).

```
C:\Users\Administrator\.jdks\openjdk-20.0.1\bin\java.exe
Bruteforce
Оптимальний набір
0 2 2
0 0 3
Максимальний прибуток
240
Час виконання: 0.07854 ms.

Process finished with exit code 0
```

Рисунок 2.9 – Результат програми методу повного перебору мовою JAVA

Нижче наведений результат програмного коду на мові Java (додаток Д), який реалізує розв’язання задачі про завантаження транспорту жадібним алгоритмом (рис. 2.10).

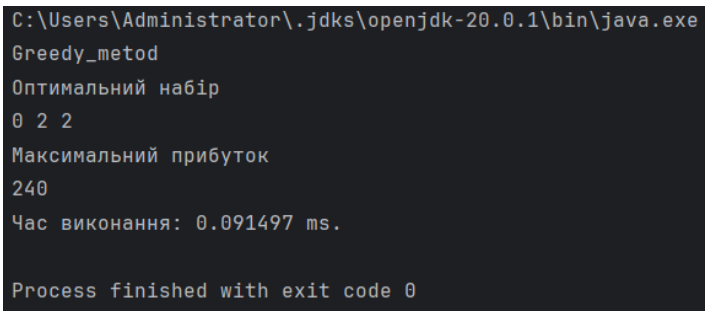


Рисунок 2.10 – Результат програми жадібним методом мовою JAVA

2.6 Розв'язання задачі в умовах статистичної невизначеності

Проблему детермінованого завантаження вже вирішено, але на практиці вага та прибуток від транспортування предметів можуть змінюватися в певному діапазоні. Тому актуально вирішувати проблеми в умовах невизначеності. Найпростішим типом невизначеності є статистична невизначеність, яка може бути задана функцією розподілу або типом і параметрами розподілу. У цьому випадку можна очікувати, що оптимальні плани будуть різними для окремих реалізацій випадкових наборів параметрів.

Для вирішення задачі про стохастичне завантаження візьмемо вихідні дані (вартість транспортування) підпорядковані нормальному закону розподілу з відхиленням 10%, 30%, 50%, 70% та 90% від детермінованих значень.

Вхідну матрицю C визначимо за нормальним законом, використовуючи функцію normrnd(MU, SIGMA) з пакету Statistics Toolbox програмного середовища MATLAB [11].

Функція normrnd(MU, SIGMA) призначена для генерації псевдовипадкових чисел за нормальним законом для кожної пари параметрів: MU – математичне сподівання, SIGMA – середньоквадратичне відхилення. Для кожного випадку відхилення необхідно знайти SIGMA. Нижче наведені значення матриці C для кожного варіанту розриву.

– матриця C для 10% відхилення :

$$C = [normrnd(100,0.9), normrnd(40,0.6), normrnd(80,0.3)];$$

– матриця C для 30% відхилення :

$$C = [normrnd(100,2.7), normrnd(40,1.8), normrnd(80,0.9)];$$

– матриця C для 50% відхилення :

$$C = [normrnd(100,4.5), normrnd(40,3), normrnd(80,1.5)];$$

– матриця C для 70% відхилення :

$$C = [normrnd(100,6.3), normrnd(40,4.2), normrnd(80,2.1)];$$

– матриця C для 90% відхилення :

$$C = [normrnd(100,8.1, normrnd(40,5.4, normrnd(80,2.7)].$$

Програма буде знаходити оптимальний план завантаження транспорту предметами, які дають максимальний прибуток цільової функції (рис. 2.11).

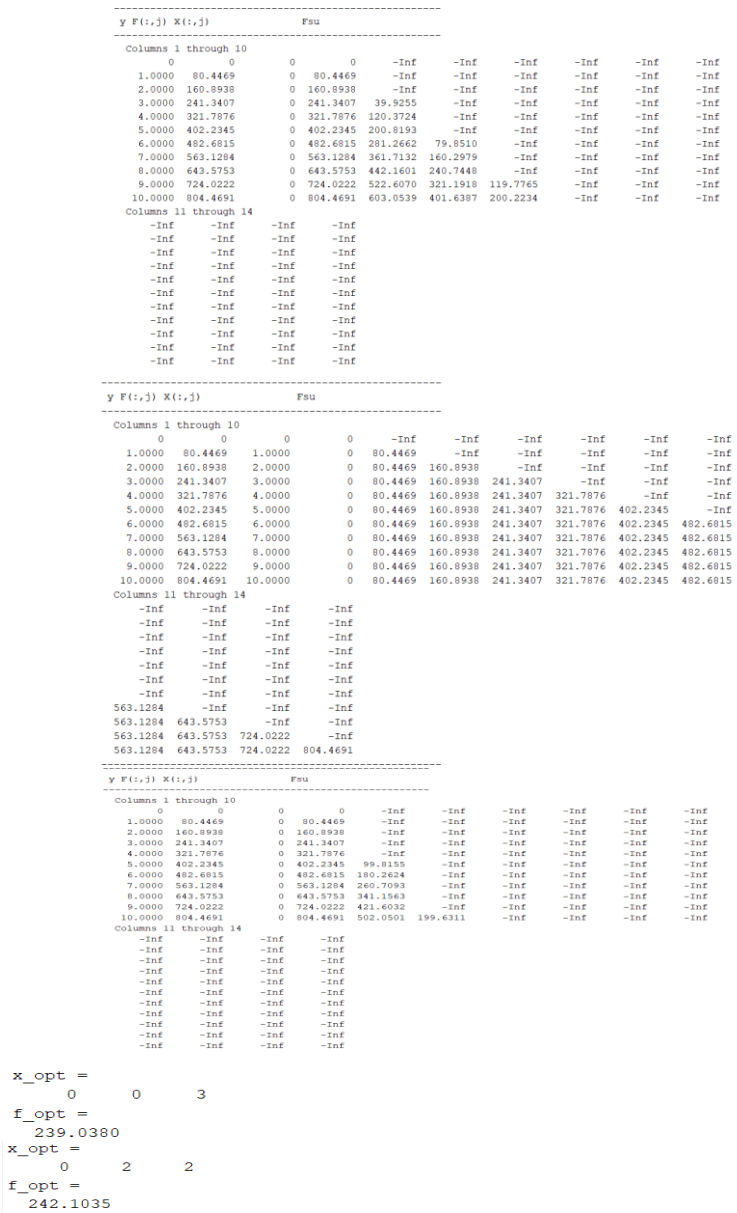


Рисунок 2.11 – Результат запуску програми методу динамічного програмування для 10% відхилення

Для кожного із варіантів розподілу було запущено програмний код 5000 разів. Для кожного випадку у нас було отримано два розв’язки, а для 90% відхилення з’явився додатковий оптимальний набір (2 0 0). Це показує, що при зміні ціни на перевезення, кількість оптимальних наборів може змінюватися. Розподіл прибутків для всіх цих варіантів проілюстровано для кожного розподілу на діаграмах «ящик з вусами», які були побудовані за допомогою функціонального блоку boxplot мови програмування R [16].

По довжині ящиків зображено міжквартильний розмах вибірки для кожного із оптимальних наборів. Середня лінія (медіана) зображує середнє

значення вибірки. Вуса показують повний розмах даних. На кінцях вусів зображено максимальне та мінімальне значення вибірки для кожного із відповідей.

Результати наведені на рисунках 2.12-2.16 (див. додатки Е-Л).

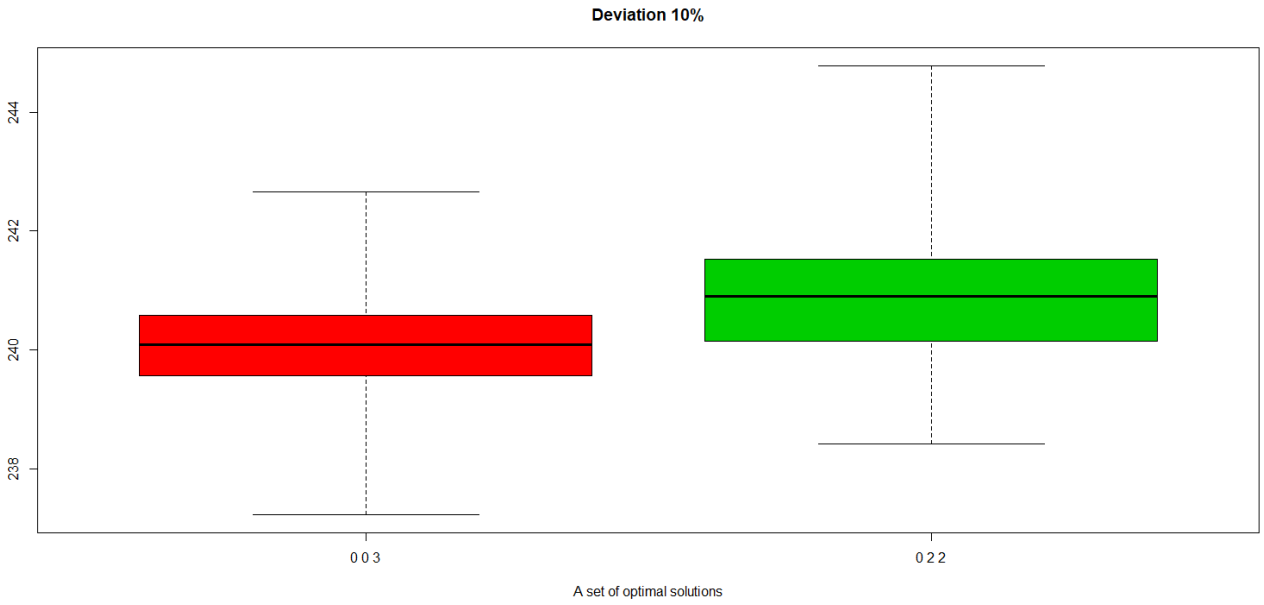


Рисунок 2.12 – Результати запусків програмного коду для 10% відхилення

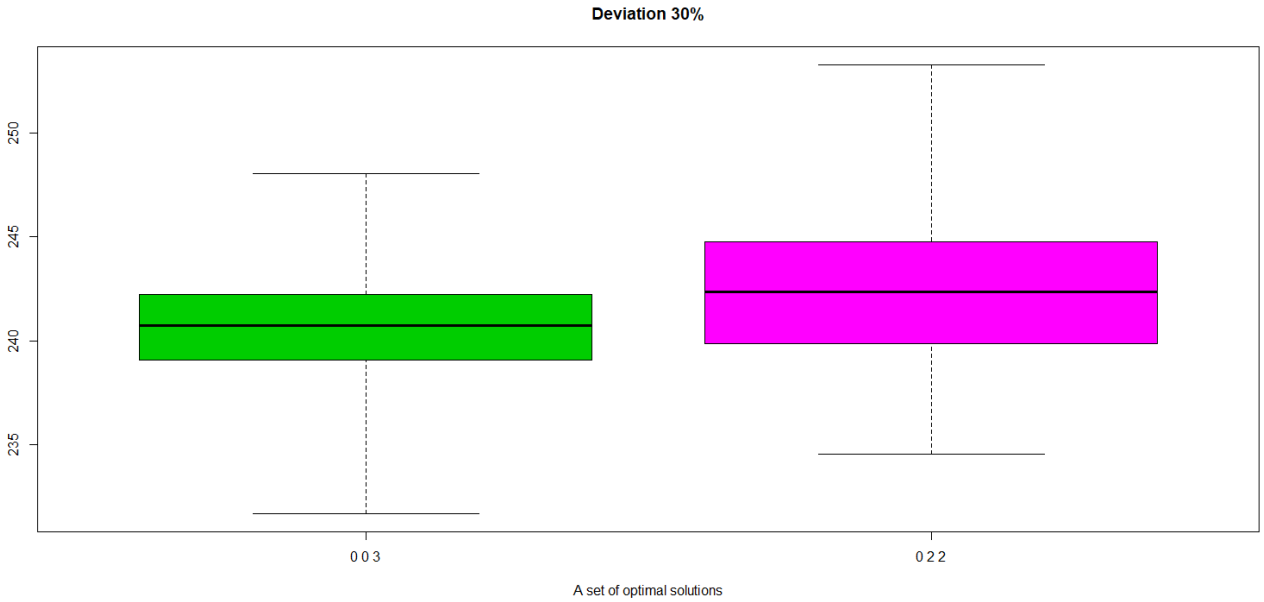


Рисунок 2.13 – Результати запусків програмного коду для 30% відхилення

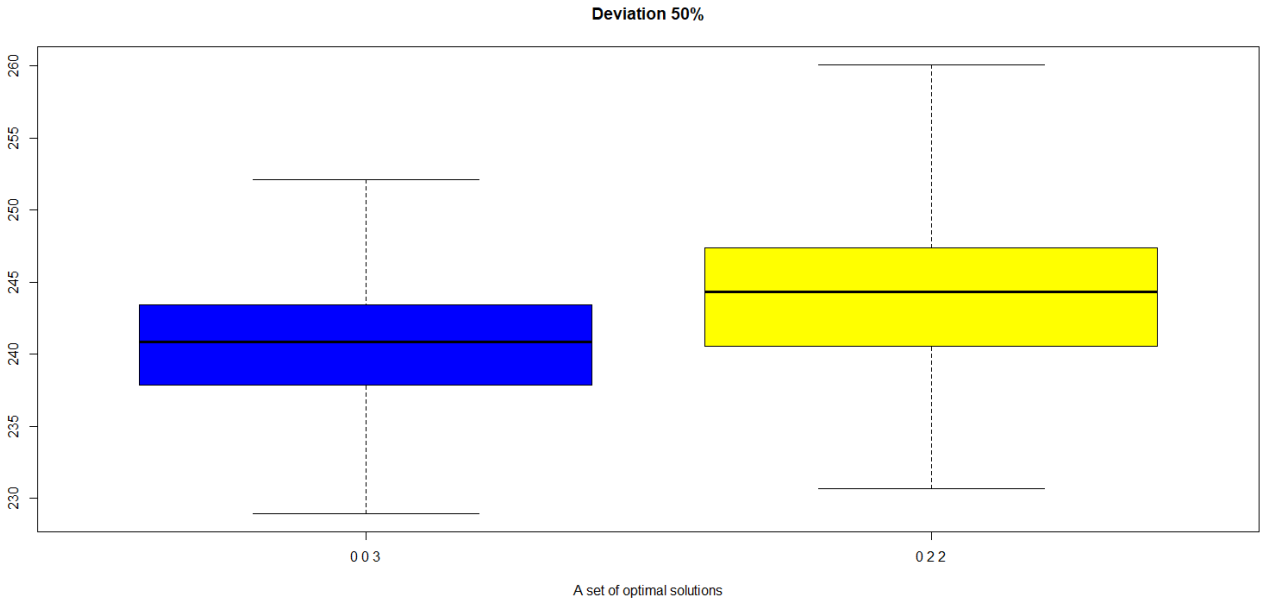


Рисунок 2.14 – Результати запусків програмного коду для 50% відхилення

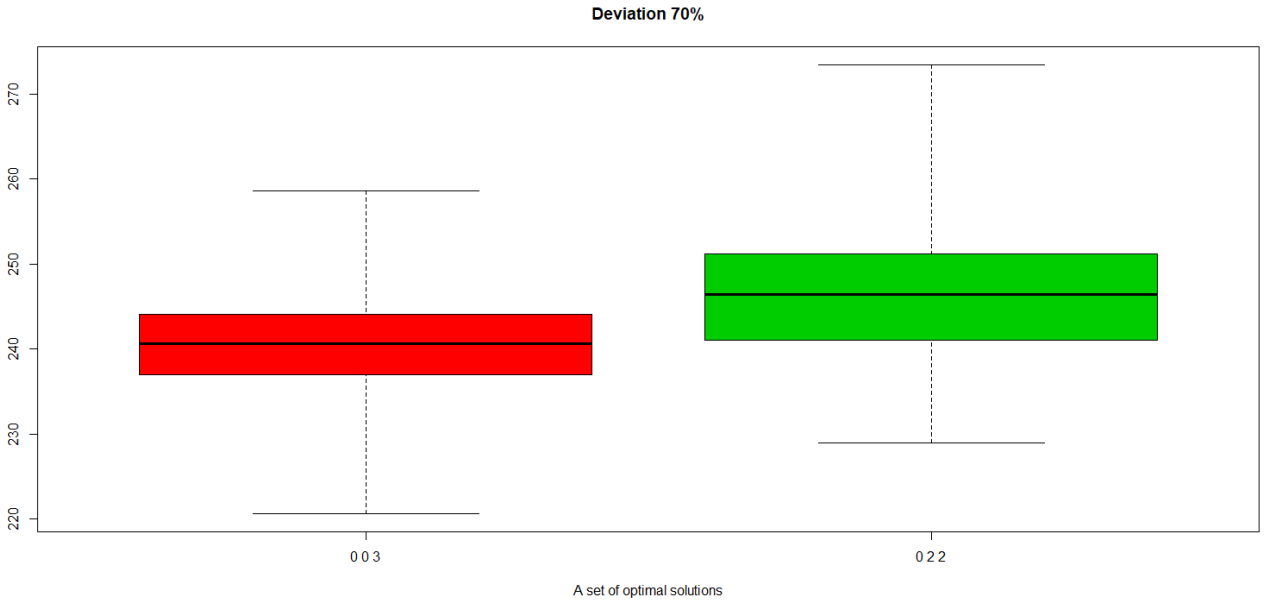


Рисунок 2.15 – Результати запусків програмного коду для 70% відхилення

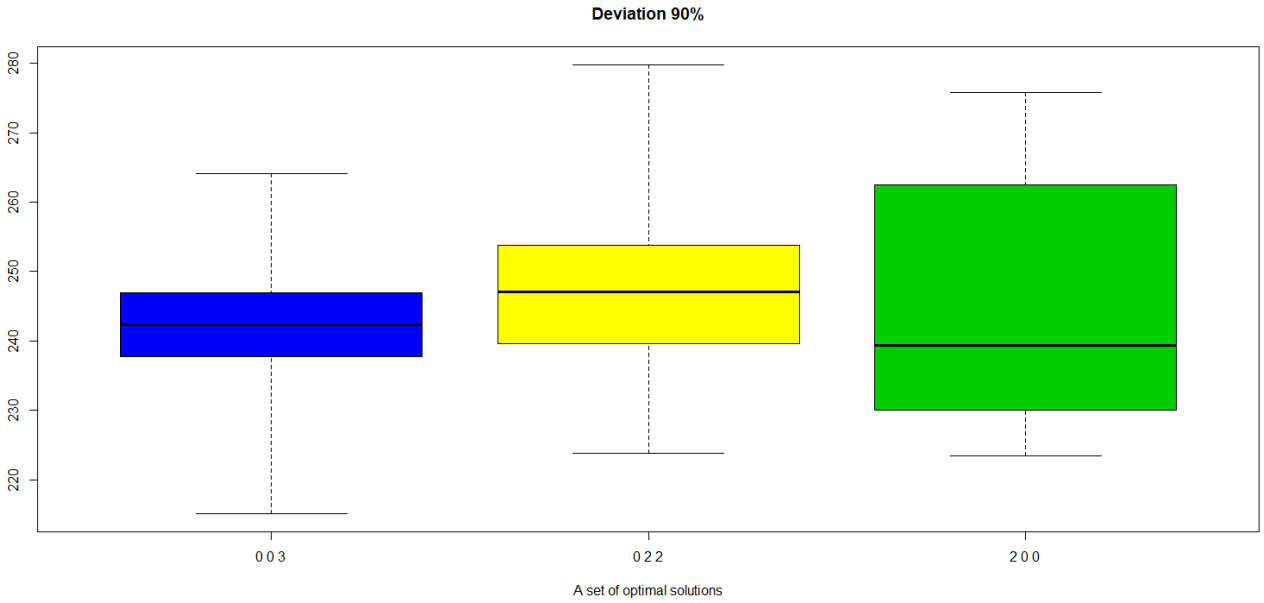


Рисунок 2.16 – Результати запусків програмного коду для 90% відхилення

2.7 Критерії оптимальності та аналіз результатів

Для того, щоб прийняти остаточне рішення, який план дає максимальний прибуток для п’яти випадків розподілу, потрібно задати критерії оптимальності.

Сформулюємо наступні критерії:

– середньоарифметичний прибуток;

– ймовірність того, що буде отримано прибуток, більше ніж 240 одиниць.

Аналіз результатів наведений у таблиці 2.5.

Таблиця 2.5 – Аналіз результатів

Результати					
Набір	10%	30%	50%	70%	90%
	Середнє арифметичне	Середнє арифметичне	Середнє арифметичне	Середнє арифметичне	Середнє арифметичне
003	240,0873	240,5994	240,6173	240,6142	242,2458
022	**240,8895**	**242,5702**	**244,1673**	**246,4055**	**247,4360**
200	-	-	-	-	244,8285

Як бачимо в таблиці 2.5 для всіх випадків розподілу за двома критеріями найкращим набором є набір (0, 2, 2), тому що він для кожного із критеріїв і для

кожного із розподілів дає найбільше середнє арифметичне значення та максимальну ймовірність того, що прибуток буде більшим за 240. Якщо ми будемо брати інші критерії, то можливо у нас зміниться оптимальний план.

ВИСНОВКИ

У кваліфікаційній роботі було зроблено наступне:

- отримано ручний розв’язок задачі про завантаження транспорту двома методами: метод перебору даних і метод динамічного програмування;
- розроблено програмний код для розв’язування детермінованої задачі та задачі в умовах статистичної невизначеності методом перебору даних, жадібним алгоритмом та методом динамічного програмування в середовищах MATLAB і JAVA;
- запропоновано різні варіанти критеріїв оптимальності;
- отримано та проаналізовано розв’язок задачі з вхідними детермінованими значеннями, розподіленими за нормальним законом.

ПЕРЕЛІК ПОСИЛАНЬ

1. Лежнев А. В. Динамічне програмування в економічних задачах: навчальний посібник Київ : БІНОМ. Лабораторія знань. 2010. 176 с.

2. Кремер Н. Ш., Путко Б. А., Тришин И. М., Фридман М. Н. Исследование операций в экономике. Москва : Изд-во Юрайт, 2018. 438 с.

3. Новицький І. В., Ус С. А. Сучасна теорія керування : навч. посіб. Дніпро : НГУ, 2017. 263 с.

4. Холявін І. І. Математичне програмування та економіко-математичні методи. Навч. посібник для студентів економ. вузів. Гатчина, 2009. 301 с.

5. Кучма М. І. Математичне програмування: приклади та задачі. Львів : Новий світ-2000, 2020. 344 с.

6. Канцедал С.А. Конструювання й дослідження алгоритмів рішення задачі про рюкзак. *Автомобільний транспорт*. 2015. № 36. С. 154-160.

7. Івченко І. Ю. Математичне програмування : навчальний посібник. Київ : Центр учбової літератури, 2007. 232 с.

8. Зайцев В. Г. Теорія керування економічними й технологічними процесами: Навч. посіб. Дніпро : РВВ ДНУ, 2005. 88 с.

9. Хорстманн К., Корнелл Г. Java. Библиотека профессионала. Киев, 2007. 896 с.

10. Horstmann C. Java SE 8 for the Really Impatient. Addison-Wesley Professional, 2014. 240 p.

11. Freeman E., Robson E. Java: A Beginner’s Guide. McGraw-Hill Education, 2019. 728 p.

12. Ranganathan K. Java EE 8 for Beginners. Packt Publishing, 2018. 378 p.

13. Креневич А. П. Алгоритми і структури даних. Київ : ВПЦ «Київський Університет», 2021. 200 с

14. Кривий С. Вступ до методів створення програмних продуктів Київ : НаУКМА, 2018. 450 с.

15. Гоблик Н. М., Гоблик В. В. MATLAB в інженерних розрахунках. Комп’ютерний практикум. Львів : Львівська політехніка, 2020. 192 с.

16. Гнатюк В. Вступ до R на прикладах. Харків : ХНЕУ, 2010. 101 с.

ДОДАТОК А

Програмний код методу перебору даних в середовищі MATLAB

```
format compact, clc
b = 10;
disp('Price:')
c = [100 40 80]
disp(' ')
a = [5 2 3];
n = length(a);
equation = b + 1; % equation - Кол-во состояний
upr = 1 + floor(b / min(a));
x_minimum = zeros(1, n);
x = x_minimum;
x_maximum = floor(b ./ a);
f_optimal = -inf;
i = n;
while 1
   if x(i) <= x_maximum(i)
      if a * x' <= b
         f = c * x';
if f_optimal < f
f_optimal = f;
x_optimal = x;
elseif f == f_optimal
      f_optimal
      x_optimal = [x_optimal; x]
end
      end
   i = n;
else
   x(i) = x_minimum(i);
i = i - 1;

if i <= 1e-5 , break, end
end
   x(i) = x(i) + 1;
end
disp('  ')
disp('answer:')
x_optimal, f_optimal
```

ДОДАТОК Б

Програмний код динамічного методу в середовищі MATLAB

```
format compact, clc

Price = [100 40 80]
disp(' ')
disp('Answer:')
Weight = [ 5 2 3 ];
b = 10;
n = length(Price);
equation = b + 1;
upr = 1 + floor(b / min(Weight));
C = -inf * ones(upr, n);
for u = 1 : upr
   for i = 1 : n
      temp = Weight(i) * (u -1);
      if temp <= b % если предметы помещаются в самолет
         C(u, i) = temp;
         R(u, i) = Price(i) * (u -1);
      end
   end
end
index_C = find(C > b);
C(index_C) = -inf * ones(1, length(index_C));
R(index_C) = -inf * ones(1, length(index_C));

X = -inf * ones(equation, n+1);
F = X;
F(:, n +1) = zeros(equation, 1);
for i = n : -1 : 1
   F_OPT = -inf * ones(equation, upr);
   for s = 1 : equation
      for u = 1 : upr
         s1 = s - C(u,i);
         if 1 <= s1 & s1 <= equation
            F_OPT(s, u) = R(u, i) + F(s1, i +1);
         end
      end
   end
   [fs, index] = max(F_OPT');
   F(:,i) = fs';
```

```
    X(:,i) = index';
disp('  ')
disp([blanks(5), 'y', blanks(5) 'F', blanks(5),'X', blanks(15), 'F_OPT'])
end
x_optimal = zeros(1, n);
s = equation;
for i = 1 : n
    x_optimal(i) = X(s, i);
    s = s - C(x_optimal(i), i);
end

x_optimal = x_optimal - 1
f_optimal = F(equation, 1)
```

ДОДАТОК В

Програмний код методу динамічного програмування мовою JAVA

```
package ru.arhiser.knapsack;
import java.util.ArrayList;


public class Dynamic_metod {
   public static void main(String[] args) {
      int[] Weights = {5, 2, 3};
      int[] Prices = {100, 40, 80};
      int count = Weights.length;
      int MaxWeight = 10;
      int[][] A;
      A = new int[count + 1][];
      for (int i = 0; i < count + 1; i++) {
         A[i] = new int[MaxWeight + 1];
      }
      for (int k = 0; k <= count; k++) {
         for (int s = 0; s <= MaxWeight; s++) {
            if (k == 0 || s == 0) {
               A[k][s] = 0;
            } else {
               if (s >= Weights[k - 1]) {
                  A[k][s] = Math.max(A[k - 1][s], A[k - 1][s - Weights[k - 1]] +
Prices[k - 1]);
               } else {
                  A[k][s] = A[k - 1][s];
               }
            }
         }
      }
long startTime = System.currentTimeMillis();
      ArrayList<Integer> result = new ArrayList<>();
      traceResult(A, Weights, count, MaxWeight, result);
      System.out.println("Оптимальний зміст:");
      for(Integer integer : result) {
        System.out.println("Максимальний прибуток:");
        System.out.println(integer);
      }
   }
```

```
    private static void traceResult(int[][] A, int[] weight, int k, int s,
ArrayList<Integer> result) {
        if (A[k][s] == 0) {
            return;
        }
        if (A[k -1][s] == A[k][s]) {
            traceResult(A, weight, k - 1, s, result);
        } else {
            traceResult(A, weight, k - 1, s - weight[k - 1], result);
            result.add(0, k);
        }
        long timeElapsed = endTime - startTime;
        System.out.println((“Час виконання” +timeElapsed);

    }
}
```

ДОДАТОК Г

Програмний код методу повного перебору даних мовою JAVA

```
package ru.arhiser.knapsack;

public class povniy_perebor {
   public static void main(String[] args) {
      int[] Weights = {5, 2, 3};
      int[] Prices = {100, 40, 80};
      int MaxWeight = 10;
      long count = 2L << Weights.length;
      int MaxPrice = 0;
      long MaxState = 0;
      for (long state = 0; state < count; state++) {
         int Price = statePrice(State, Prices);
         int Weight = stateWeight(State, Weights);
         if (Weight <= MaxWeight) {
            if (MaxPrice < Price) {
               MaxPrice = Price;
               MaxState = State;
            }
         }
      }
long startTime = System.currentTimeMillis();
      System.out.println("Оптимальний зміст:");
      long PoverOfTwo = 1;
      for (int i = 0; i < Weights.length; i++) {
         if ((PoverOfTwo & MaxState) > 0) {
        System.out.println("Максимальний прибуток:");
        System.out.println(i + 1);
         }
         poverOfTwo <<= 1;
      }
   }
long timeElapsed = endTime - startTime;
      System.out.println((“Час виконання” +timeElapsed);

   private static int stateWeight(long state, int[] Weights) {
      long PoverOfTwo = 1;
      int Weight = 0;
      for (int i = 0; i < Weights.length; i++) {
```

```
            if ((PoverOfTwo & State) != 0) {
                Weight += Weights[i];
            }
            PoverOfTwo <<= 1;
        }
        return Weight;
    }

    private static int statePrice(long state, int[] Prices) {
        long PoverOfTwo = 1;
        int Price = 0;
        for (int i = 0; i < Prices.length; i++) {
            if ((PoverOfTwo & State) != 0) {
                Price += Prices[i];
            }
            PoverOfTwo <<= 1;
        }
        return Price;
    }
}
```

ДОДАТОК Д

Програмний код жадібного методу мовою JAVA

```
package ru.arhiser.knapsack;
import java.util.ArrayList;
public class Greedy {

    public static void main(String[] args) {
        int[] Weights = {5, 2, 3};
        int[] Prices = {100, 40, 80};
        int MaxWeight = 10;
        ArrayList<Integer> indexes = new ArrayList<>();
        ArrayList<Integer> result = new ArrayList<>();
        int ResultWeight = 0;

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < Weights.length; i++) {
            indexes.add(i);
        }
        while (!indexes.isEmpty()) {
            int MaxValue = Prices[indexes.get(0)];
            int MaxIndex = indexes.get(0);
            for (int i = 1; i < indexes.size(); i++) {
                if (MaxValue < Prices[indexes.get(i)]) {
                    MaxValue = Prices[indexes.get(i)];
                    MaxIndex = indexes.get(i);
                }
            }

            ResultWeight += Weights[MaxIndex];
            if (ResultWeight > MaxWeight) {
                break;
            }
            result.add(MaxIndex);
            indexes.remove(MaxIndex);
        }

        System.out.println("Оптимальний зміст:");
        for (Integer integer : result) {
System.out.println("Максимальний прибуток:");
            System.out.println(integer + 1);
```

```
        }
  long endTime = System.currentTimeMillis();

        long timeElapsed = endTime - startTime;
        System.out.println("Час виконання" +timeElapsed);
    }
}
```

ДОДАТОК Е

Програмний код для побудови графіків «ящик з вусами» мовою R для 10% відхилення від вхідних даних

```
data003<-read.table("003.csv", sep=";", dec=",", header=TRUE)
data022<-read.table("022.csv", sep=";", dec=",", header=TRUE)

data003
data022

boxplot(c(data003,data022), main = "Deviation 10%", col = c ("2", "3"), names = c("0
0 3", "0 2 2"), xlab="A set of optimal solutions", range = 0)
```

ДОДАТОК Ж

Програмний код для побудови графіків «ящик з вусами» мовою R для 30% відхилення від вхідних даних

```
data003<-read.table("003.csv", sep=";", dec=",", header=TRUE)
data022<-read.table("022.csv", sep=";", dec=",", header=TRUE)

data003
data022

boxplot(c(data003,data022), main = "Deviation 30%", col = c ("3", "6"), names = c("0 0 3", "0 2 2"), xlab="A set of optimal solutions", range = 0)
```

ДОДАТОК И

Програмний код для побудови графіків «ящик з вусами» мовою R для 50% відхилення від вхідних даних

```
data003<-read.table("003.csv", sep=";", dec=",", header=TRUE)
data022<-read.table("022.csv", sep=";", dec=",", header=TRUE)

data003
data022

boxplot(c(data003,data022), main = "Deviation 50%", col = c ("4", "7"), names = c("0 0 3", "0 2 2"), xlab="A set of optimal solutions", range = 0)
```

ДОДАТОК К

Програмний код для побудови графіків «ящик з вусами» мовою R для 70% відхилення від вхідних даних

```
data003<-read.table("003.csv", sep=";", dec=",", header=TRUE)
data022<-read.table("022.csv", sep=";", dec=",", header=TRUE)

data003
data022

boxplot(c(data003,data022), main = "Deviation 70%", col = c ("2", "3"), names = c("0 0 3", "0 2 2"), xlab="A set of optimal solutions", range = 0)
```

ДОДАТОК Л

Програмний код для побудови графіків «ящик з вусами» мовою R для 90% відхилення від вхідних даних

```
data003<-read.table("003.csv", sep=";", dec=",", header=TRUE)
data022<-read.table("022.csv", sep=";", dec=",", header=TRUE)
data200<-read.table("200.csv", sep=";", dec=",", header=TRUE)

data003
data022
data200

boxplot(c(data003,data022,data200), main = "Deviation 90%", col = c ("4", "7","3"),
names = c("0 0 3", "0 2 2", "2 0 0"), xlab="A set of optimal solutions", range = 0)
```