МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

СЕКЦІЯ ІКТ

ВИПУСКНА РОБОТА

на тему:

«Комп'ютерний порівняльний аналіз алгоритмів розв'язання задач цілочисельної оптимізації»

Завідувач

випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Шаповалов С. П.**

Студентка групи ІНз-73-9С **Мештер Т. О.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної і вечірньої форм навчання
Кафедра комп'ютерних наук

Затверджую___________

Зав. кафедрою Довбиш А.С.

"_____" ____________2021 р.

ЗАВДАННЯ

до випускної роботи

Студентки п'ятого курсу, групи ІНз-73-9С спеціальності "Комп'ютерні науки" заочної форми навчання Мештер Тетяни Олександрівни

Тема:«Комп'ютерний порівняльний аналіз алгоритмів розв'язання задач цілочисельної оптимізації»

Затверджена наказом по СумДУ

№________ от ___________2021 р.

Зміст пояснювальної записки: 1) аналітичний огляд застосувань структур даних ; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, математичних моделей і алгоритмів, що використовуються для рішення поставленого завдання; 5) розробка інформаційного й програмного забезпечення; 6) аналіз результатів моделювання.

Дата видачі завдання "_____"____________2021 р.

Керівник випускної роботи __________ Шаповалов С. П.

Завдання прийняв до виконання ____________ Мештер Т. О.

РЕФЕРАТ

Записка: 36 стор., 10 рис., 3 табл., 2 додатки, 11 джерел інформації.

Об'єкт дослідження — задачі цілочисельної оптимізації.

Мета роботи — комп'ютерний порівняльний аналіз алгоритмів розв'язання задач цілочисельної оптимізації.

Методи дослідження — математичне моделювання, комп'ютерна реалізація алгоритмів на ЕОМ.

Результати — розроблено інформаційне та програмне забезпечення комп'ютерного порівняльного аналізу алгоритмів розв'язання задач цілочисельної оптимізації. Проведено огляд алгоритмів та обрані алгоритми для порівняння між собою. Виконано комп'ютерну реалізацію за допомогою алгоритмічної мови програмування Java.

ЦІЛОЧИСЕЛЬНА ОПТИМІЗАЦІЯ, ПОРІВНЯЛЬНИЙ
КОМП'ЮТЕРНИЙ АНАЛІЗ, АЛГОРИТМ МОНТЕ_КАРЛО
МОВА ПРОГРАМУВАННЯ JAVA

ЗМІСТ

ВСТУП.. 5

1 АНАЛІЗ ПРОБЛЕМИ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ 6

1.1Аналітичній огляд алгоритмів рішення задачі цілочисельної оптимізації.. 7

1.2 Постановка задачі.. 11

2 МАТЕМАТИЧНА ПОСТАНОВКА ЗАВДАЧІ ТА ВИБІР МЕТОДУ ЇЇ РІШЕННЯ. 12

2.1 Математична постановка 12

2.2 Алгоритм Монте-Карло.. 13

2.3 Адаптація алгоритму Монте-Карло для рішення задач цілочиседьної рптимізації 16

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. ТЕСТОВІ РОЗРАХУНКИ……………………………………………………18

3.1 Комп’ютерна реалізація алгоритмів та основні її складові 18

3.2 Порівняльний аналіз тестових прикладів... 21

ВИСНОВКИ... 25

СПИСОК ЛІТЕРАТУРИ.. 26

ДОДАТОК А 27

ДОДАТОК Б.. 32

ВСТУП

Велика кількість проблем з оптимальним рішенням у науковій, технічній та державній сферах наштовхується як на нелінійну динаміку системи, так і на цілочисельність рішення [1-7]. Ці проблеми вирішення призводять до змішаного цілого нелінійного програмування (MINLP) проблеми, що поєднують комбінаторну складність оптимізації над дискретними варіантами здатні набори з проблемами роботи з нелінійними функціями.

На перший погляд здається, що рішення задач цілочисельного оптимізації можна знайти, округляючи рішення задачі, отриманої з вихідної, відмовившись від вимоги цілочисельності керованих змінних. Однак можна навести приклади, коли побудоване таким чином рішення буде далеким від оптимального або навіть неприпустимим [1-3].

Можна надати дві основні причини використання цілочисельних змінних при моделюванні оптимізаційних задач:

1. Цілісні змінні представляють величини, які можуть бути лише цілими. Наприклад, неможливо побудувати 3,7 машини.
2. Цілочисельні змінні представляють рішення (наприклад, чи включати спеціаліста для виконання роботи), тому вони повинні приймати лише значення 0 або 1.

Ці міркування часто трапляються на практиці, тому цілочисельна оптимізація може використовуватися в багатьох задачах інфомаційно-комунікаційних технологій.

Серед відомих алгоритмів, які розвинені для рішень задач цілочисельної оптимізації відомі алгоритм Гоморі та метод гілок на меж, але вони вирішують тільки задачі в лінійній постановці. Для загального разу єдиного способу чи алгоритму не існує, тому проблема його пошуку є актуальною.

1 АНАЛІЗ ПРОБЛЕМИ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

Математична енциклопедія надає таке означення; цілочисельне програмування – розділ математичного програмування, що досліджує задачі оптимізації (максимізації або мінімізації) функції декількох змінних, пов'язаних рівняннями або нерівностями та задовольняючих умову цілочисельності.

Умова цілочисельності змінних формально відображає: а) фізичну неподільність об'єктів (наприклад, при розміщенні підприємств або виробництві виробів); б) скінченність множини допустимих варіантів, на якому проводиться оптимізація (наприклад, множини перестановок в задачах упорядкування); в) наявність логічних умов, виконання або невиконання яких веде до зміни виду цільової функції і обмежень задачі.

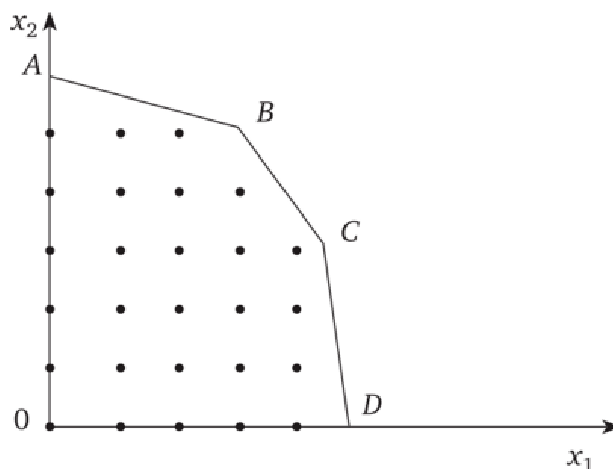

Рисунок 1.1 – Ілюстрація вибору рішення в області серед цілих чисел

Задачі цілочисельної оптимізації відносять до більш широкого класу дискретних задач, в яких змінні приймають значення з деякої дискретної множини. До таких завдань відносяться, наприклад, завдання планування перевезення вантажу в контейнерах, обсяги яких приймають дискретні значення, завдання планування виробництва за умов, що використовуються агрегати мають певні дискретні потужності. Такого роду задачі включають також і завдання булевого програмування, в яких змінні приймають тільки два значення: 0 або 1 (є булеві змінними).

До моделей цілочисельної оптимізації за таким загальним означенням можна додати такі відомі задачі, як задача комівояжера й задача про призначення.

1.1 Аналітичній огляд алгоритмів рішення задачі цілочисельної оптимізації

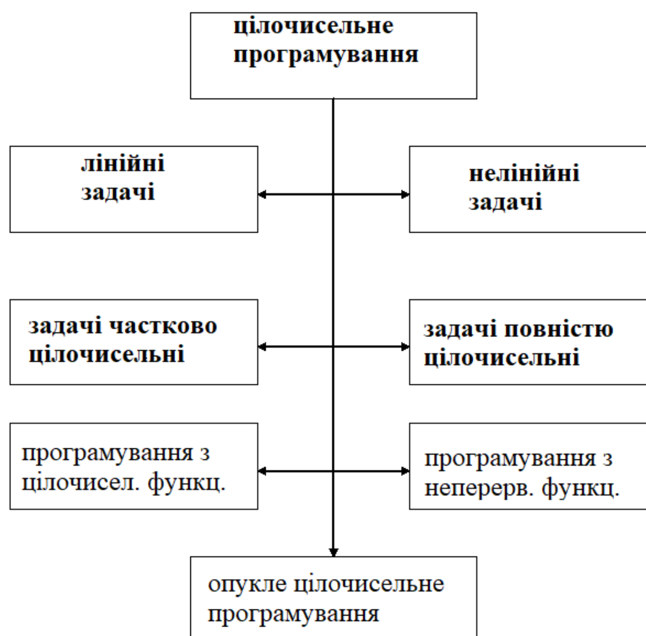

Рисунок 1.2 – класифікація задач цілочисельного програмування

Для знаходження рішень завдань цілочисельного програмування алгоритми умовно можна розподілити на три основних групи:

- методи відтинання;
- комбінаторні методи;
- наближені методи.

Коротко оглянемо алгоритми цих груп.

Методи відтинання.

Суть методу полягає в тому, що спочатку вирішується завдання програмування без умови цілочисельності. Якщо в результаті рішення отримаємо відповідь, що задовольняє умові цілочисельності, то задача вирішена. В іншому випадку до обмежень, що задані в умові задачі додаємо нове обмеження, що задовольняє наступним властивостям:

1. воно повинно бути лінійним;
2. воно повинно відтинати знайдене оптимальне нецілочисельне рішення;
3. воно не повинно відтинати жодного цілочисельного плану.

Такі обмеження називаються правильними відтинаннями.

Для ілюстрації дії методу відтинання розглянемо рішення задачі. Представленої в [10] й заданої математичною моделлю.

$$y(\bar{x}) = x_1 + 2x_2 \to \max_{x \in \Omega}$$

$$\Omega : \begin{cases} f_1 = 4x_1 - 2x_2 \leq 11 \\ f_2 = -x_1 + 3x_2 \leq 9 \\ f_3 = x_1 + 2x_2 \geq 2 \\ x_j \geq 0, j = 1,2 \\ x_j - цілі \end{cases}$$

(1. 1)

Так як поставлене завдання має тільки дві змінні для оптимізації, то побудуємо область рішень графічно (див. рис. 1.3). За алгоритмом, розв'язуємо задачу без умови цілочисельності. Розв'язок, як не важко зрозуміти є точкою перетину прямих f_1 та f_2 , але це рішення не є цілочисельним: x* = 5,1; y* = 4.7 [10].

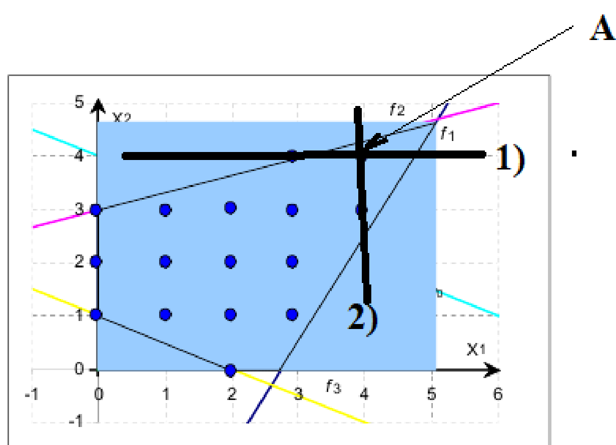

Рисунок 1.3 – До рішення задачі методом відтинання

Проведемо два правильних віддинання: 1) та 2) , що представлені на рис. 1.3 прямими. Одержимо два обмеження $x \leq 4$ та $y \leq 4$, що додаємо до заданої системи обмежень.

Розв'язуючи нове завдання з тією ж самою функцією цілі, але новими обмеженнями одержимо рішення в точці A, що вказано на рис. 1.3

Координати цієї точки x=4; y=4 й є шуканим рішенням. F(4,4) = 12.

Найвідомішим алгоритмом з цієї когорти є алгоритм Гоморі.

Комбінаторні методи.

Метод гілок і меж - один з комбінаторних методів вирішення завдань цілочисельного програмування. Його суть полягає в упорядкованому переборі варіантів і розгляді лише тих з них, які виявляються за певними ознаками перспективними, і відкиданні безперспективних варіантів.

Метод гілок і меж полягає в наступному: множина припустимих рішень деяким чином розбивається на підмножини, кожне з яких цим же способом розбивається ще на підмножини. Процес триває до тих пір, поки не буде знайдений оптимальний план.

Його суть полягає в тому, що множина планів розбивається на ряд підмножин. Для кожного з підмножин знаходять оцінку цільової функції по оптимуму, що представляє собою самостійну, більш просту, ніж вихідна, завдання. На основі цих оцінок вибирають «перспективні» підмножини і повторюють процедуру їх поділу на нові підмножини.

Алгоритм дії за цим методом наступний.

Нехай задача цілочисельного лінійного програмування вирішена симплекс-методом без урахування цілочисельності і відомі верхня і нижня межа для кожної цілої змінної $\alpha_j \le x_j \le \beta_j$ та нижня межа цільової функції Z_0 : $f(X) \ge Z_0$ для будь-якого плану X.

Припустимо що компонента x_i* рішення оптимального плану не задовольняє умові цілочисельності. Тоді з області допустимих рішень виключається область $[x^*_i] \le x_i \le \{x_i^*] +1$.

Початкова задача розпадеться на дві (2) і (3). У завдання (2) додається обмеження $\alpha_i \le x_i \le \{x_i^*] +1$, а в завдання (3) додається обмеження $[x^*_i] \le x_i. \le \beta_i$

Вирішуємо ці завдання. В результаті список завдань може або розширитися, або зменшитися. Якщо в результаті вирішення завдань (2) або (3) нецілочисельне оптимальний план, при якому $f(x^*) \le Z_0$, то це завдання виключається. Якщо $f(x^*) > Z_0$, то з даного завдання формуються дві нові.

Якщо отримане рішення x* задовольняє умові цілочисельності і $f(x^*) \le Z_0$, то значення Z_0 виправляється і за величину Z_0 приймається оптимум лінійної функції отриманого оптимального цілочисельного плану.

Процес триває до тих пір, поки весь список завдань не буде вичерпаний, тобто всі завдання будуть вирішені.

Для ілюстрації методу розглянемо задачу.

$Z = 350x_1+150x_2 \rightarrow max$; При обмеженнях

$$\begin{aligned} 25x_1 + 10x_2 &\le 100, \\ 40x_1 + 20x_2 &\le 190, \end{aligned} \qquad (1.2)$$

x_1, x_2 – цілі.

За алгоритмом, розв'язуємо задачу без умови цілочисельності. Розв'язок, як не важко зрозуміти є : $x_1^* = 1$; $x_2^* = 7,5$ а $Z^* = 1475$.

x_2^* - змінна, що задовольняє цілочисельності. Тоді за алгоритмом займемся змінною x_2^*.Одержимо дві задачі.

2) до моделі (1.2) додається нерівність $x_2 \le 7$, й рішення знайдемо нової задачі: $x_1 = 1,2$; $x_2 = 7$; Z=1470.

3) до моделі (1.2) додається нерівність $x_2 \ge 8$, й рішення знайдемо нової задачі: $x_1 = 0,75$; $x_2 = 8$; Z=1462,5.

Оскільки цілочисельного розв'язку не одержали, продовжуємо процес.

Розгалуження задачі показано на рис. 1.4.

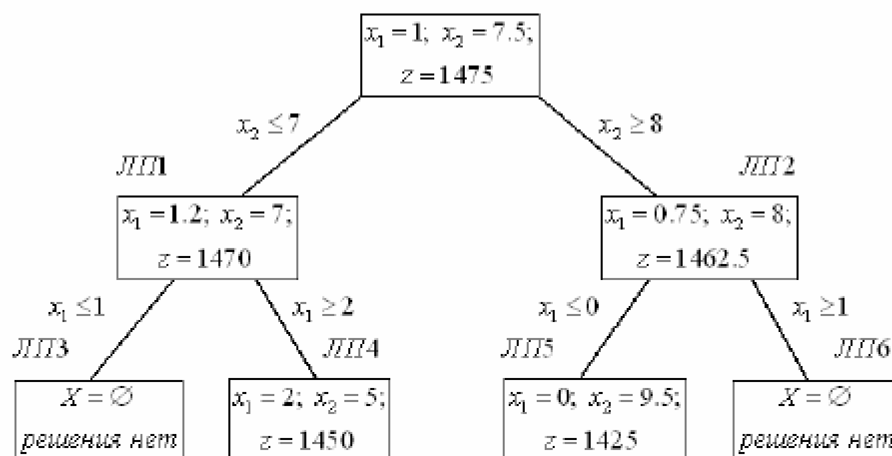

Рисунок 1.4 – Розгалудження задач при методі гілок иа меж

Остаточне рішення $x_1 = 2$; $x_2 = 5$; Z=1450.

Наближені методи.

Наближені методи діляться умовно на два види: методи, які використовують випадковий процедури пошуку і методи, які використовують евристичні прийоми.

Серед випадкових процедур, виділяється метод Монте-Карло, що є нашою метою до застосування. Тому його ми розглянемо в розділі 2.

Оскільки завдання цілочисельного програмування відносяться до класу NP-повних, іноді наближені методи є єдиними, що приводять до їх рішення.

Тому генетичні алгоритми, алгоритми мурашиної колонії та інші все більше й більше знаходять застосування в цілочисельній оптимізації.

1.2 Постановка задачі

Поставимо наступне завдання для дослідження.

Розробити комп'ютерну програму, яка буде розв'язувати проблеми цілочисельної оптимізації використовуючи метод Монте-Карло. Для реалізації використати тестові завдання як лінійного так і нелінійного програмування.

Для досягнення мети знайти рішення підзадач:

1. *Дослідити адаптацію алгоритму Монте-Карло для рішення цілочисельної оптимізації.*
2. *Провести порівняльний аналіз алгоритмів на тестових завданнях.*
3. *На основі проведених досліджень зробити висновки застосовності алгоритму в практичному застосуванні.*

2 МАТЕМАТИЧНА ПОСТАНОВКА ЗАВДАЧІ ТА ВИБІР МЕТОДУ ЇЇ РІШЕННЯ

2.1 Математична постановка

До задач цілочисельної оптимізації (цілочисельного програмування) відносяться такі, результати рішення яких (шукані значення параметрів оптимізації) повинні бути цілими числами.

Такого роду проблеми можуть виникати як в лінійному програмуванні. Тоді вони мають математичну модель представлення:

Максимізувати $\sum_{j=1}^{n} c_j \, x_j$

в заданих обмеженнях $\sum_{j=1}^{n} a_{ij} \, x_j < b_i$

$x_j \geq 0$, x_j - цілі.

Для проблеми в нелінійному та загальному разі математичну постановку можна задати наступне

Поставимо загальну математичну постановку задачі нелінійного програмування:

Максимізувати(мінімізувати) функцію цілі F

$\mathbf{F = f(x_1, x_2, x_3, \ldots, x_n) \rightarrow max(min),}$ (2. 1)

в якій керовані змінні $(\mathbf{x_1, x_2, x_3, \ldots, x_n})$ вибираються з області допустимих рішень **D**, що задається сукупністю обмежень

$$\begin{cases} G(\,(\mathbf{x1, x2, x3, \ldots, xn}) \geq \mathbf{0}, \\ Q(\,(\mathbf{x1, x2, x3, \ldots, xn}) \leq \mathbf{0}, \\ \mathrm{W}(\mathbf{x1, x2, x3, \ldots, xn}) \geq \mathbf{0} \end{cases} \quad (2.\ 2)$$

При додаткових обмеженнях $x_j \geq 0$, x_j - цілі, для всіх $j = 1, 2, \ldots, n$.

Розв'язанням (рішенням) задачі цілочисельної оптимізації є пошук в області допустимих рішень такого набору значень $\mathbf{X(x_1, x_2, x_3, \ldots, x_n)}$ який приведе функцію цілі до шуканого екстремуму.

Оптимізація - це вибір найкращого варіанта з багатьох можливостей. Якщо критерій вибору відомий і варіантів небагато, то рішення може бути знайдено за допомогою перебору та порівняння всіх варіантів. Однак, на практиці буває частіше, що число можливих варіантів настільки велике, що повний перебір практично неможливо виконати. У таких випадках приходить формулювання задачі

з мови математики та застосування спеціальних методів пошуку оптимального рішення, тобто методи оптимізації.

2.2 Алгоритм Монте-Карло

Алгоритм Монте-Карло відноситься до класу наближених обчислювальних алгоритмів, які покладаються на багаторазові випадкові вибірки процесів або явищ для отримання числових результатів. Основна концепція полягає у використанні випадковості для вирішення проблем, які в принципі можуть бути детермінованими. В основу алгоритму покладено запуск рулетки випадкових чисел для рішення проблем.

Привабливість алгоритму Монте-Карло в його прозорості та нескладності в застосуваннях. Він часто використовуються для рішення проблем у фізичних та математичних моделюваннях і є найбільш корисними, коли важко або неможливо використовувати інші підходи. Іноді складний, важкий для програмування алгоритм рішення тієї чи іншої проблеми легко розбивається елегантним застосуванням при її рішенні алгоритму Монте-Карло.

Методи Монте-Карло в основному використовуються в трьох класах задач: [4-7] оптимізація, чисельне інтегрування та генерування черпань із розподілу ймовірностей.

Рисунок 2.1 – Ілюстрація методу для пошуку площі складної фігури

Алгоритм Монте-Карло розрізняється в деталях, але як правило має єдину центральну концепцію:

1) Визначається область можливих входів.
2) Визначається ціль дослідження або модель процесу.
3) Генеруються вхідні дані випадковим чином.
4) Перевіряється допустимість згенерованих даних і якщо так, то визначається на них функція цілі чи результати проходження процесу.
5) Результати «правильних» генерацій співвідносяться між собою та обирається серед них найкращий.

Є два важливих моменти в зв'язку з цією концепцією.

Перший момент. Якщо область можливих входів має значні розміри, то важливо щоб генерації випадковим чином рівномірно її покривали. Якщо точки не розподілені рівномірно, то наближення буде поганим.

Другий момент. Використання алгоритму Монте-Карло вимагає генерації великої кількості випадкових чисел (може бути мільйони генерацій). Це стимулювало розробку генераторів псевдовипадкових чисел, які використовувались набагато швидше, ніж таблиці випадкових чисел, які раніше використовувались для статистичної вибірки. Кожна мова програмування високого рівня має генератори псевдовипадкових чисел і працює над їх поліпшенням.

Цьому важливому аспекту привернули велику увагу, наприклад на мові Java. Існує кілька способів як згенерувати випадкове число. Розглянемо генерацію чисел за допомогою Math.random ().

У бібліотеці класів Java є пакет java.lang, у якого є клас Math, а у класу Math є метод random () (див. рис. 2.2).

При кожному виклику Math.random () с допомогою спеціального алгоритму (за певною інструкцією) генерується випадкове число. А оскільки все-таки існує невелика ймовірність передбачити, яке ж число буде згенеровано алгоритмом, такі числа прийнято називати не випадковими, а псевдовипадковими.

За замовчуванням Math.random () генерує випадкові речові числа з проміжку [0; 1), тобто від нуля включно до 1 виключно. Тому виникають питання яким чином згенерувати числа з заданого проміжку, при цьому вони повинні ще бути цілими й невід'ємними (за умовою постановки див. п. 2.1).

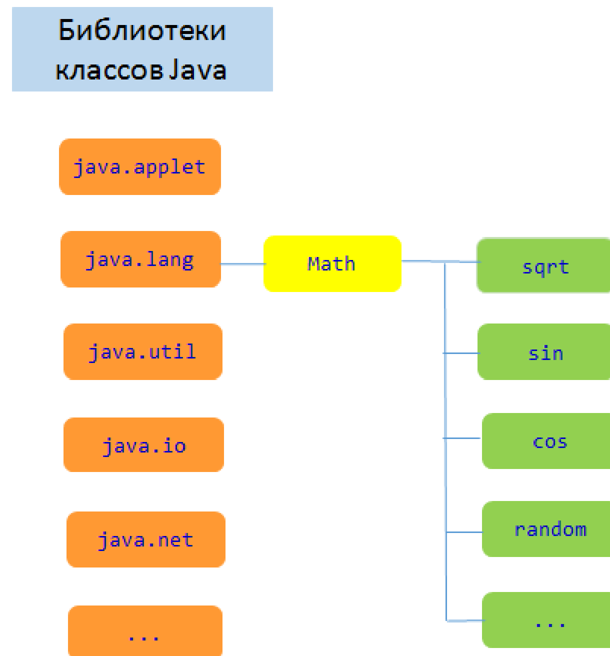

Рисунок 2.2 – Ієрархія бібліотеки класів Java

Формально представимо діапазон генерування [a; b). Тобто нижня межа a, верхня b. Тоді для генерації дійсних чисел в заданому діапазоні потрібно записати наступне:

(Math.random () * (b-a)) + a.

Тепер повернемось до завдання, як згенерувати цілочисельне значення?

Покажемо це на прикладі.

Поставимо завдання згенерувати число в інтервалі [0; 2]. Відразу помічаємо, що після 2 дужка квадратна, а не кругла. Це означає, що нас цікавить, щоб діапазон включав в себе число 2. Алгоритмічно це можна провести наступне:

```
public class Test {
        public static void main(String[] args){
                int a = (int) ( Math.random() * 3 );
        }
}
```

По-перше, ми збільшили діапазон [0, 1) утричі. Тоді Math.random() * 3 генерує число в діапазоні [0, 3). А операція int – округлює число до цілого. Отже ми одержимо число саме з інтервалу [0; 2]. Отже проблема розв’язана.

Моделювання Монте-Карло насправді є випадковими експериментами, у тому випадку, якщо результати цих експериментів недостатньо відомі. Моделювання Монте-Карло, як правило, характеризується багатьма невідомими параметрами, багато з яких важко отримати експериментально.

Методи моделювання Монте-Карло не завжди вимагають справді випадкових чисел, щоб бути корисними. У багатьох найкорисніших техніках використовуються детерміновані псевдовипадкові послідовності, що дозволяє легко тестувати та повторно запускати моделювання.

Єдиною якістю, яка зазвичай необхідна для хорошого моделювання, є те, щоб псевдовипадкова послідовність виглядала "досить випадковою" у певному сенсі. Що це означає, залежить від програми, але зазвичай вони повинні пройти ряд статистичних тестів. Перевірка рівномірного розподілу чи слідування за іншим бажаним розподілом, коли враховується досить велика кількість елементів послідовності, є одним з найпростіших і найпоширеніших. Слабкі співвідношення між послідовними вибірками також часто є бажаними / необхідними.

2.3 Адаптація алгоритму Монте-Карло для рішення задачі цілочисельної оптимізації

Отже, нехай задано в загальному разі задачу цілочисельної оптимізації (2.1), (2.2). Адаптуємо алгоритм Монте-Карло для рішення поставленої задачі.

Застосування генетичного алгоритму, описаному в п. 2.2 буде в наступному.

1. *Задаємо кількість ітерацій N (генерацій) для розіграшу* $\mathbf{X(x_1, x_2, x_3, \ldots, x_n)}$. *Кожна координата вектору має нижню та верхню межі генерації й генерується в своїх межах (див. п. 2.2 -генерація цілочисельних рішень в межах).*
2. *Генеруємо на кожній ітерації набори* $\mathbf{X(x_1, x_2, x_3, \ldots, x_n)}$. *Кожну генерацію перевіряємо на предмет входження в область допустимих*

рішень. С цією метою, використовуємо сито нерівностей (2.2). Ті генерації, що задовольняють (2.2) залишаються для подальших дій, інші відкидаються.

3. *Обчислюємо на кожному кроці генерації функцію цілі для тих генерацій, що пройшли відбір за допомогою (2.2) й «кращих» (тих що надають функції цілі більших значень) зберігаєм на цьому кроці, разом зі значеннями функції цілі для них.*
4. *На кожній ітерації (генерації) порівнюємо нові рішення з тими що збереглися на минулому кроці й залишаємо кращі з них.*
5. *Процес закінчуємо при вичерпаності числа генерацій або по вичерпаності часу, що відпущений на алгоритм Монте-Карло.*

Проведемо тестові дослідження по застосовності створеного методу Рішення задач цілочисельної оптимізації.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТОВІ РОЗРАХУНКИ

3.1 Комп’ютерна реалізація алгоритмів та основні її складові

Для комп’ютерної реалізації проекту було обрано мову програмування Java, що є сьогоденні однією з найпопулярніших мов програмування.

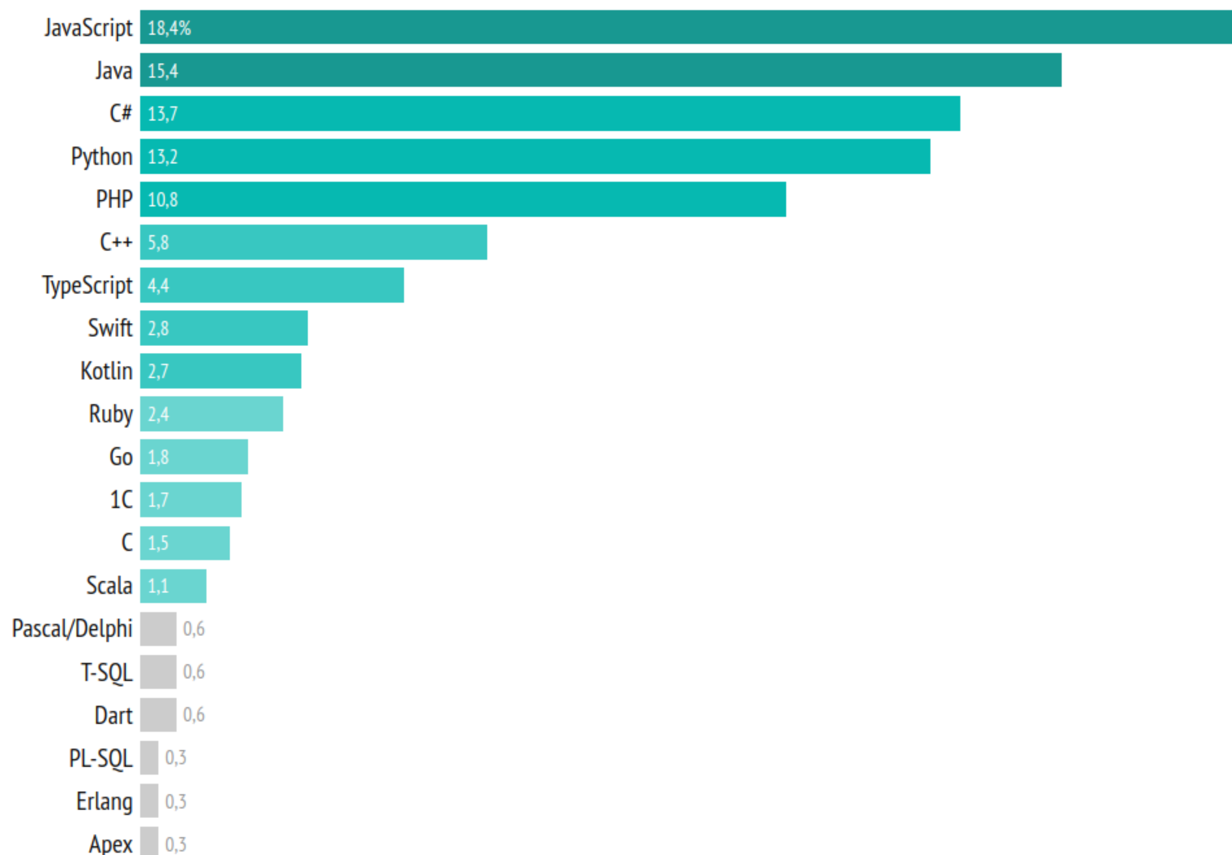

Рисунок 3.1 – Рейтинг мов програмування за 2020 рік

Java - це високорівнева, об'єктно-орієнтована мова програмування на основі класів, яка розроблена з якомога меншою залежністю реалізації.

Це мова програмування загального призначення, призначена для того, щоб розробники програм могли писати один раз, запускати їх де завгодно, що означає, що скомпільований код Java може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції. Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM), незалежно від базової архітектури комп'ютера[8-9].

Java був протестований програмістами в самих різних сферах: від кишенькових комп'ютерів до інтерактивного телебачення. Зараз він найбільш затребуваний в таких напрямках, як

- ❖ веб-розробка (масштабні бізнес-проекти);
- ❖ програми для ПК (десктопний софт);
- ❖ комп'ютерні ігри (наприклад, Minecraft);
- ❖ додатки для мобільних пристроїв (ОС Android);
- ❖ наукові дослідження і розробки;
- ❖ промисловий програмінг.

При реалізації алгоритму Менте-Карло для рішення задач цілочисельної оптимізації на мові Java, основним елементом комп'ютерної реалізація є генерація цілочисельного масиву. Масив заповнювався цілими числами які генерувалися

```
/* Генерує випадкове число */
static int getRandomNumber(double min, double max) {
    return (int) ((Math.random() * (max - min)) + min);
```

В цьому фрагменті min і max – це нижня та верхня межа інтервалів для генерування цілих чисел.

Для знаходження максимальних можливих меж було створено розділ в програмі

```
}
/* Знаходить максимальні можливі значення кожної змінної */
static int[] getCoordinateMaxValues(int[][] limits) {
    int[] coordinateMaxValues = new int[limits[0].length - 1];
    int[][] coordinateMaxPossibleValues = new int[limits.length][limits[0].length - 1];
    for (int i = 0; i < limits.length; i++) {
        for (int j = 0; j < limits[0].length - 1; j++) {
            coordinateMaxPossibleValues[i][j] = limits[i][limits[0].length - 1] / limits[i][j];
        }
    }
```

```
    for (int i = 0; i < coordinateMaxPossibleValues[0].length; i++) {
      int value = Integer.MAX_VALUE;
      for (int j = 0; j < coordinateMaxPossibleValues.length; j++) {
        if (coordinateMaxPossibleValues[j][i] <= value) {
          value = coordinateMaxPossibleValues[j][i];
        }
      }
      coordinateMaxValues[i] = value;
    }
    return coordinateMaxValues;
  }
```

Мінімальні значення для генерацій в принципі можна встановити нульовими.

Такого роду алгоритмізація знаходження меж генерування не завжди вдається. Це вдалося провести при комп'ютерній реалізації тесту №1. В загальному разі межі для генерування встановлюються дослідниками вручну.

Комп'ютерну адаптація алгоритму Монте-Карло для рішення задач цілочисельної оптимізації можна представити фрагментарно.

1. Задаємо функцію цілі.
2. Задаємо систему обмежень.
3. Встановлюємо межі генерування для керованих змінних.
4. Задаємо число цілочисельних генерацій.
5. Генеруємо цілочисельні рішення.
6. Ті, що задовольняють систему обмежень залишаємо й підставляємо їх в функції цілі. Ті, в яких функція цілі більша запам'ятовуємо до наступної генерації та перевірки функції цілі.
7. Після того як число генерацій буде вичерпано, знаходим рішення.

3.2 Порівняльний аналіз тестових прикладів

Тест №1 [умова задачі з 11]. Для виготовлення 3 видів виробів A, B і C використовується токарне, фрезерне, зварювальне та шліфувальне обладнання. Затрати часу надані в таблиці 3.1. В ній же вказаний фонд часу кожного обладнання, а також прибуток від реалізації одного виробу по кожному з видів.

Потрібно визначитись, яку кількість виробів й якого виду потрібно виготувати підприємству, щоб прибуток від реалізації був максимально можливий для даного випадку.

Таблиця 3.1 Таблиця вхідних даних для тесту №1.

Тип обладнання	Затрати часу на обробку 1 виробу виду			Загальний фонд часу, ч
	A	B	C	
Фрезерне	2	4	5	120
Токарне	1	8	6	280
Зварювальне	7	4	5	240
Шліфувальне	4	6	7	360
Прибуток, грн	10	14	12	

Складемо математичну модель задачі. Очевидно, що функція цілі Z – є прибуток від реалізації виробів й потрібно її максимізувати. Отримаємо, позначивши вироби A за x_1, B – x_2 та C – за x_3. Очевидно, що вироби не можуть бути дробовими, тобто $x_i \geq 0$ і $\in Z$.

$$Z = f(x_1, x_2, x_3) = 10x_1+14x_2+12x_3 \to max \qquad (3.1)$$

Система обмежень (3.2) має вигляд

$$\begin{cases} 2x_1 + 4x_2 + 5x_3 \leqslant 120, \\ x_1 + 8x_2 + 6x_3 \leqslant 280, \\ 7x_1 + 4x_2 + 5x_3 \leqslant 240, \\ 4x_1 + 6x_2 + 7x_3 \leqslant 360 \end{cases}$$

Адаптація алгоритму Монте-Карло для цього тестового прикладу надана в додатку А. Для аналізу результатів рішення на предмет досяжності точності було проведено ряд іспитів з різною кількістю числа генерацій.

Таблиця 3.2 Результати досліджень за тестом №1

Кількість реалізацій	Керовані змінні			Функ. цілі Z
	x_1	x_2	x_3	
100	24	11	5	454
500	21	18	1	474
1000	16	21	0	454
5000	21	17	2	472
10000	23	17	1	480
50000	24	18	0	492
100000	24	18	0	492

Графік функції цілі в залежності від кількості генерацій має вид

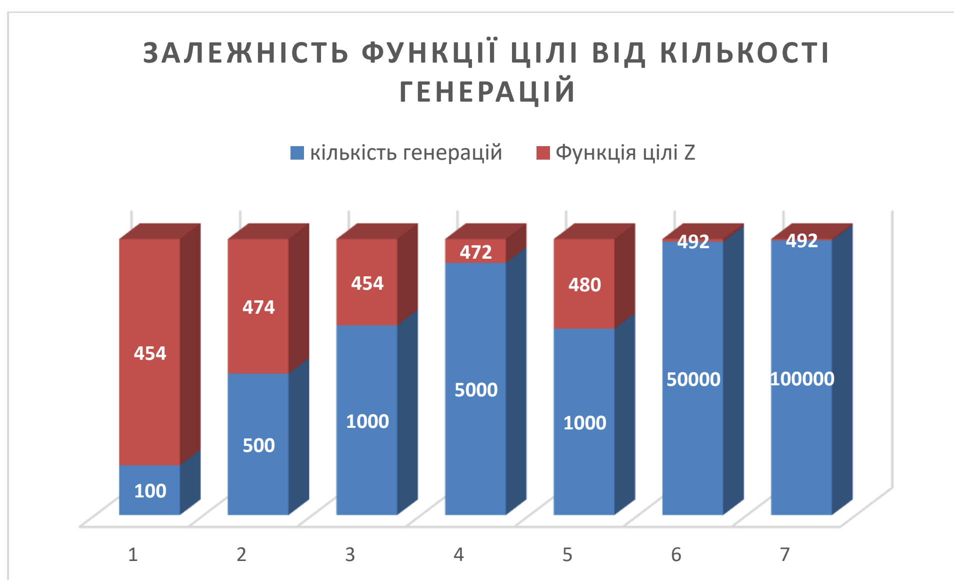

Рисунок 3. 2 – Гістограма показу Z=f(N), де N – кількість генерацій

Тестування вказує, що вже результати при кількості генерацій N = 50000 та N=100000 є ідентичними й досягли точних значень.

На самому разі ця задача має рішення Z = 492 при x_1 =24; x_2 =18 x_3=0.

Отже, алгоритм Монте-Карло одержав точне рішення, правда кількість генерацій для цього має значний розмір.

Тест №2 [умова задачі з 11]. Знайти максимальне значення функції

$$Z = f(x_1, x_2) = x_2 - x_1^2 + 6x_1 \rightarrow \max \qquad (3.3)$$

при обмеженнях (3.4)

$$\begin{cases} 2x_1+3x_2\leqslant 24, \\ x_1+2x_2\leqslant 15, \\ 3x_1+2x_2\leqslant 24, \end{cases}$$

$$x_2\leqslant 4,$$

$$x_1,\quad x_2\geqslant 0.$$

В даному разі ми отримали задачу нелінійного програмування, так як функція цілі є нелінійною.

Для отримання рішення та застосування алгоритму Монте-Карло створимо додаток В.

Для ілюстрації рішення наведемо графічне представлення завдання (див. рис. 3.2

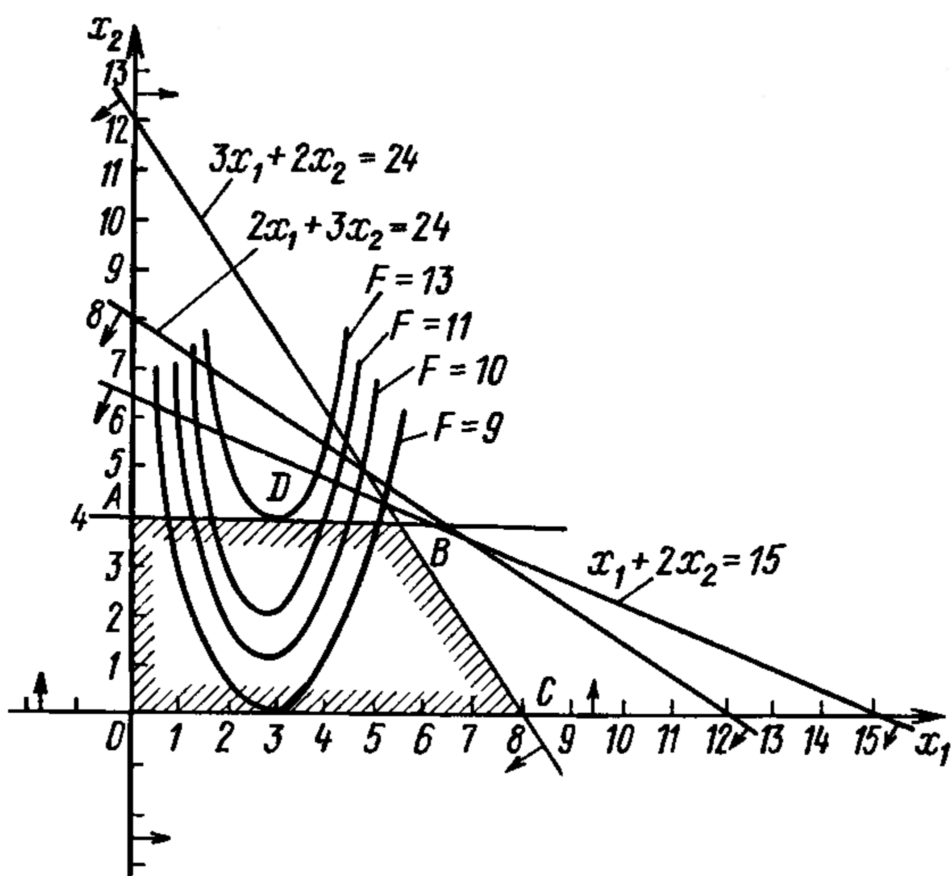

Рисунок 3. 3 – Ілюстрація тесту №2.

Результати рішення тесту №2 представлені в табл. 3.3

Таблиця 3.3 Результати досліджень за тестом №2

Кількість реалізацій	x_1	x_2	Z
100	3	4	13
500	3	4	13
1000	3	4	13
5000	3	4	13
10000	3	4	13
50000	3	4	13
100000	3	4	13

В результаті ми отримали рішення миттєво вже при незначній кількості генерацій N =100. Це рішення збіглось з точним рішенням.

ВИСНОВКИ

У випускній роботі досліджено алгоритми та методи розв'язання задач цілочисельної оптимізації. Проведений аналіз дозволив обрати для вирішення постановки задачі алгоритм Монте-Карло, що входить до наближених методів.

За результатами проведених досліджень можна зробити такі висновки.

1. Алгоритм Монте-Карло є універсальним алгоритмом та легко адаптується до розв'язання задач цілочисельної оптимізації.
2. Точність рішень за цим алгоритмом визначається кількістю координат, що генеруються в шуканому векторі рішень.
3. Тестові розрахунки показали, що результати за алгоритмом Монте-Карло сходяться до точних рішень при малій кількості компонент рішення.
4. Якщо число компонент рішення завелике й кожна з компонент має достатньо великий діапазон допустимих варіантів, то алгоритм Монте-Карло надає тільки наближеного рішення, що залежить від кількості випадкових генерацій.

СПИСОК ЛІТЕРАТУРИ

1. Математичні методи дослідження операцій: підручник/ Є. А. Лавров, Л. П. Перхун, В. В. Шендрик та ін. – Суми: Сумський державний університет, 2017. – 212 с.
2. Таха Хэмди А. Исследование операций/ Таха Хэмди А. – Москва: Вильямс, 2016. – 912 с.
3. Ефимова Г. А., Страхов Е. М. Целочисленное программирование. – Одесский национальный университет имени И.И. Мечникова, 2014. – 39 с.
4. Mazhdrakov M. The Monte Carlo Method: Engineering Applications/ Mazhdrakov M. , Benov D. , Valkanov N. – ACMO Academic Press, 2018. – 250 p.
5. Шаповалов С. П. Застосування методу Монте-Карло в обчислювальних моделях // МА:: ІМА – 2020. Матеріали та програма науково-технічної конференції СумДУ. 2020. - с. 84.
6. Gobet E., Monte-Carlo Methods and Stochastic Processes. From Linear to Non-Linear. – Chapman and Hall/CRC, London, 2020. – 336 p.
7. Lemire D. Fast Random Integer Generation in an Interval, 2019. - 15 p. [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/1805.10941.pdf
8. Friesen J. Data structures and algorithms in Java, Part 1: Overview, 2017. - [Електронний ресурс] – Режим доступу до ресурсу: https://www.infoworld.com/article/3215112/java-101-datastructures-and-algorithms-in-java-part-1.html
9. Lafore R. Data Structures and Algorithms in Java . - Sams Publishing, 2020. – 800 p.
10. Білогурова Г.В., Самойленко М.І. Математичне програмування: Конспект лекцій. – Х.: ХНАМГ, 2009. – 72 с.
11. Акулич И. Л. Математическое программирование в примерах и задачах. – М.: Высшая школа, 1986. – 319 с.

ДОДАТОК А

Тест №1

```
import javafx.util.Pair;
import java.io.IOException;
import java.util.Arrays;


public class MonteCarloLinear {
    public static void main(String[] args) throws IOException {
        /* Система обмеженнь */
        int[][] limits = new int[][]{
                {2, 4, 5, 120},
                {1, 8, 6, 280},
                {7, 4, 5, 240},
                {4, 6, 7, 360}};
        /* Коефіцієнти цільової функції */
        int[] objectiveFunctionCoefficients = new int[]{10, 14, 12};
        /* Можливі варіанти кількості точок */
        int[] amountOfPoints = new int[]{100, 500, 1000, 5000, 10000, 50000,
100000};
        for (int i = 0; i < amountOfPoints.length; i++) {
            Pair<int[], Integer> result = monteCarloMethod(limits,
objectiveFunctionCoefficients, amountOfPoints[i]);
```

```
            System.out.println(amountOfPoints[i] + " " + Arrays.toString(result.getKey())
+ " " + result.getValue());
        }
    }

    /* Метод Монте Карло */
    static Pair<int[], Integer> monteCarloMethod(int[][] limits, int[]
objectiveFunctionCoefficients, int amountOfPoints) {
        /* Масив координат точок, які відповідають оптимальному значенню */
        int[] optimalCoordinates = new int[0];
        /* Масив максимальних можливих значень кожної змінної */
        int[] coordinatesMaxValues = getCoordinateMaxValues(limits);
        /* Оптимальне значення цільової функції */
        int objectiveFunctionValue = Integer.MIN_VALUE;
        for (int i = 0; i < amountOfPoints; i++) {
            /* Генеруємо точку */
            int[] coordinates = new int[limits[0].length - 1];
            for (int j = 0; j < limits[0].length - 1; j++) {
                coordinates[j] = getRandomNumber(0, coordinatesMaxValues[j]);
            }
            /* Перевіряємо чи потрапляє точка у область */
            if (isCoordinatesInArea(limits, coordinates)) {
                /* Обчислюємо значення цільової функції */
```

```
            int res = calculateObjectiveFunction(objectiveFunctionCoefficients,
coordinates);
            /*  Оновлюємо оптимальне значення та координати у разі необхідності
*/
            if (res > objectiveFunctionValue) {
               objectiveFunctionValue = res;
               optimalCoordinates = coordinates;
            }
         }
      }
      return new Pair<>(optimalCoordinates, objectiveFunctionValue);
   }
   /* Знаходить максимальні можливі значення кожної змінної */
   static int[] getCoordinateMaxValues(int[][] limits) {
      int[] coordinateMaxValues = new int[limits[0].length - 1];
      int[][] coordinateMaxPossibleValues = new int[limits.length][limits[0].length -
1];
      for (int i = 0; i < limits.length; i++) {
         for (int j = 0; j < limits[0].length - 1; j++) {
            coordinateMaxPossibleValues[i][j] = limits[i][limits[0].length - 1] /
limits[i][j];
         }
      }
```

```
    for (int i = 0; i < coordinateMaxPossibleValues[0].length; i++) {
        int value = Integer.MAX_VALUE;
        for (int j = 0; j < coordinateMaxPossibleValues.length; j++) {
            if (coordinateMaxPossibleValues[j][i] <= value) {
                value = coordinateMaxPossibleValues[j][i];
            }
        }
        coordinateMaxValues[i] = value;
    }
    return coordinateMaxValues;
}
/* Перевіряє умови потрапляння точки в область */
static boolean isCoordinatesInArea(int[][] limits, int[] coordinates) {
    for (int i = 0; i < limits.length; i++) {
        if (!checkCriteria(limits[i], coordinates)) {
            return false;
        }
    }
    return true;
}
/* Перевіряє критерій */
static boolean checkCriteria(int[] limit, int[] coordinates) {
```

```
        return limit[0] * coordinates[0] + limit[1] * coordinates[1] + limit[2] *
coordinates[2] <= limit[3];
    }
    /* Цільова функція */
    static int calculateObjectiveFunction(int[] objectiveFunctionCoefficients, int[]
coordinates) {
        return objectiveFunctionCoefficients[0] * coordinates[0] +
objectiveFunctionCoefficients[1] * coordinates[1]
                + objectiveFunctionCoefficients[2] * coordinates[2];
    }
    /* Генерує випадкове число */
    static int getRandomNumber(double min, double max) {
        return (int) ((Math.random() * (max - min)) + min);
    }
```

}ДОДАТОК Б

Тест №2

```
import javafx.util.Pair;


import java.io.IOException;
import java.util.Arrays;


public class MonteCarloNonlinear {
   public static void main(String[] args) throws IOException {
       /* Система обмеженнь */
       int[][] limits = new int[][]{
               {2, 3, 24},
               {1, 2, 15},
               {3, 2, 24},
               {0, 1, 4}};
       /* Можливі варіанти кількості точок */
       int[] amountOfPoints = new int[]{100, 500, 1000, 5000, 10000, 50000,
100000};
       for (int i = 0; i < amountOfPoints.length; i++) {
           Pair<int[], Integer> result = monteCarloMethod(limits, amountOfPoints[i]);
           System.out.println(amountOfPoints[i] + " " + Arrays.toString(result.getKey())
+ " " + result.getValue());
       }
```

```
    }

    /* Метод Монте Карло */
    static Pair<int[], Integer> monteCarloMethod(int[][] limits, int amountOfPoints) {
        /* Масив координат точок, які відповідають оптимальному значенню */
        int[] optimalCoordinates = new int[0];
        /* Масив максимальних можливих значень кожної змінної */
        int[] coordinatesMaxValues = getCoordinateMaxValues(limits);
        /* Оптимальне значення цільової функції */
        int objectiveFunctionValue = Integer.MIN_VALUE;
        for (int i = 0; i < amountOfPoints; i++) {
            /* Генеруємо точку */
            int[] coordinates = new int[limits[0].length - 1];
            for (int j = 0; j < limits[0].length - 1; j++) {
                coordinates[j] = getRandomNumber(0, coordinatesMaxValues[j] + 1);
            }
            /* Перевіряємо чи потрапляє точка у область */
            if (isCoordinatesInArea(limits, coordinates)) {
                /* Обчислюємо значення цільової функції */
                int res = calculateObjectiveFunction(coordinates);
                /*  Оновлюємо оптимальне значення та координати у разі необхідності
*/
                if (res > objectiveFunctionValue) {
```

```
                objectiveFunctionValue = res;
                optimalCoordinates = coordinates;
            }
        }
    }
    return new Pair<>(optimalCoordinates, objectiveFunctionValue);
}

/* Знаходить максимальні можливі значення кожної змінної */
static int[] getCoordinateMaxValues(int[][] limits) {
    int[] coordinateMaxValues = new int[limits[0].length - 1];
    int[][] coordinateMaxPossibleValues = new int[limits.length][limits[0].length -
1];
    for (int i = 0; i < limits.length; i++) {
        for (int j = 0; j < limits[0].length - 1; j++) {
            if (limits[i][j] != 0) {
                coordinateMaxPossibleValues[i][j] = limits[i][limits[0].length - 1] /
limits[i][j];
            } else {
                coordinateMaxPossibleValues[i][j] = Integer.MAX_VALUE;
            }
        }
    }
```

```
    for (int i = 0; i < coordinateMaxPossibleValues[0].length; i++) {
        int value = Integer.MAX_VALUE;
        for (int j = 0; j < coordinateMaxPossibleValues.length; j++) {
            if (coordinateMaxPossibleValues[j][i] <= value) {
                value = coordinateMaxPossibleValues[j][i];
            }
        }
        coordinateMaxValues[i] = value;
    }
    return coordinateMaxValues;
}

/* Перевіряє умови потрапляння точки в область */
static boolean isCoordinatesInArea(int[][] limits, int[] coordinates) {
    for (int i = 0; i < limits.length; i++) {
        if (!checkCriteria(limits[i], coordinates)) {
            return false;
        }
    }
    return true;
}

/* Перевіряє критерій */
```

```
    static boolean checkCriteria(int[] limit, int[] coordinates) {
        return limit[0] * coordinates[0] + limit[1] * coordinates[1] <= limit[2];
    }
    /* Цільова функція */
    static int calculateObjectiveFunction(int[] coordinates) {
        return (int) (coordinates[1] - Math.pow(coordinates[0], 2) + 6 * coordinates[0]);
    }
    /* Генерує випадкове число */
    static int getRandomNumber(double min, double max) {
        return (int) ((Math.random() * (max - min)) + min);
    }
}
```