МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №7**

по дисциплине: Исследование операций

тема: Решение полностью целочисленных задач с помощью первого

алгоритма Гомори, а также методом ветвей и границ

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

Вирченко Юрий Петрович

Белгород 2024 г.

**Цель работы:** Освоить методы отсечения Гомори для полностью целочисленных задач. Изучить алгоритм этого метода. Программно реализовать этот алгоритм.

**Задания**

1. Изучить возможные постановки задач целочисленного и частично- целочисленного программирования.

2. Ознакомиться с методами решения таких задач, в частности, с методами отсечения и методом ветвей и границ.

3. Выяснить для каких задач применяется первый алгоритм Гомори. Изучить этот алгоритм и написать реализующую его программу для ПЭВМ. Изучить и программно реализовать алгоритм метода ветвей и границ. В качестве тестовых данных использовать, решенную вручную следующую задачу.

Изображение выглядит как текст, Шрифт, рукописный текст, белый

Автоматически созданное описание**Вариант 3**

**Ручной расчет**

Решим прямую задачу линейного программирования симплексным методом, с использованием симплексной таблицы.

Определим максимальное значение целевой функции при следующих условиях-ограничений.

Расширенная матрица системы ограничений-равенств данной задачи:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 1 | 0 | 0 | 93 |
| 14 | -5 | 0 | -1 | 0 | 26 |
| 2 | -9 | 0 | 0 | -1 | 18 |

Приведем систему к единичной матрице методом жордановских преобразований.

1. В качестве базовой переменной можно выбрать .

2. В качестве базовой переменной можно выбрать .

Получаем новую матрицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 1 | 0 | 0 | 93 |
| -14 | 5 | 0 | 1 | 0 | -26 |
| 2 | -9 | 0 | 0 | -1 | 18 |

3. В качестве базовой переменной можно выбрать .

Получаем новую матрицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 1 | 0 | 0 | 93 |
| -14 | 5 | 0 | 1 | 0 | -26 |
| -2 | 9 | 0 | 0 | 1 | -18 |

Поскольку в системе имеется единичная матрица, то в качестве базисных переменных принимаем X = (3,4,5).

Выразим базисные переменные через остальные:

Подставим их в целевую функцию:

или

Среди свободных членов имеются отрицательные значения, следовательно, полученный базисный план не является опорным.

Вместо переменной следует ввести переменную

Выполняем преобразования симплексной таблицы методом Жордано- Гаусса.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 |
| x3 |  | 0 |  | 1 |  | 0 |
| x1 |  | 1 |  | 0 |  | 0 |
| x5 |  | 0 |  | 0 |  | 1 |
|  |  | 0 |  | 0 |  | 0 |

Представим расчет каждого элемента в виде таблицы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| B | x1 | x2 | x3 | x4 | x5 |
| 93-(-  26\*10):-14 | 10-(-  14\*10):-14 | 3-(5\*10):-  14 | 1-(0\*10):-  14 | 0-(1\*10):-  14 | 0-(0\*10):-  14 |
| -26 : -14 | -14 : -14 | 5 : -14 | 0 : -14 | 1 : -14 | 0 : -14 |
| -18-(-26\*- 2):-14 | -2-(-14\*- 2):-14 | 9-(5\*-2):-  14 | 0-(0\*-2):-  14 | 0-(1\*-2):-  14 | 1-(0\*-2):-  14 |

Среди свободных членов bi имеются отрицательные значения, следовательно, полученный базисный план не является опорным.

Вместо переменной x5 следует ввести переменную x4.

Выполняем преобразования симплексной таблицы методом Жордано- Гаусса.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 |
| x3 | 3 | 0 | 48 | 1 | 0 | 5 |
| x1 | 9 | 1 |  | 0 | 0 |  |
| x4 | 100 | 0 | -58 | 0 | 1 | -7 |
|  | -189 | 0 |  | 0 | 0 |  |

Представим расчет каждого элемента в виде таблицы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Выразим базисные переменные через остальные:

Подставим их в целевую функцию:

или

При вычислениях значение временно не учитываем.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 48 | 1 | 0 | 5 |
| 1 | -9/2 | 0 | 0 | -1/2 |
| 0 | -58 | 0 | 1 | -7 |

Матрица коэффициентов этой системы уравнений имеет вид:

A =

**Базисные переменные** это переменные, которые входят только в одно уравнение системы ограничений и притом с единичным коэффициентом.

**Экономический смысл дополнительных переменных**: дополнительные переменные задачи ЛП обозначают излишки сырья, времени, других ресурсов, остающихся в производстве данного оптимального плана.

Решим систему уравнений относительно базисных переменных:

Полагая, что свободные переменные равны 0, получим первый опорный план:

**Базисное решение** называется допустимым, если оно неотрицательно.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 |
| x3 | 3 | 0 | 48 | 1 | 0 | 5 |
| x1 | 9 | 1 |  | 0 | 0 |  |
| x4 | 100 | 0 | -58 | 0 | 1 | -7 |
|  | 0 | 0 |  | 0 | 0 |  |

Переходим к основному алгоритму симплекс-метода.

**Итерация №0.**

**1. Проверка критерия оптимальности.**

Текущий опорный план неоптимален, так как в индексной строке находятся отрицательные коэффициенты.

**2. Определение новой базисной переменной.**

В качестве ведущего выберем столбец, соответствующий переменной x2, так как это наибольший коэффициент по модулю.

**3. Определение новой свободной переменной.**

Вычислим значения Di по строкам как частное от деления и из них выберем наименьшее:

Следовательно, 1-ая строка является ведущей.

Разрешающий элемент равен (48) и находится на пересечении ведущего столбца и ведущей строки.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 | min |
| x3 | 3 | 0 | 48 | 1 | 0 | 5 |  |
| x1 | 9 | 1 |  | 0 | 0 |  | - |
| x4 | 100 | 0 | -58 | 0 | 1 | -7 | - |
|  | 0 | 0 |  | 0 | 0 |  | 0 |

**4. Пересчет симплекс-таблицы.**

Формируем следующую часть симплексной таблицы. Вместо переменной в план 1 войдет переменная .

Строка, соответствующая переменной в плане 1, получена в результате деления всех элементов строки плана 0 на разрешающий элемент РЭ = 48. На месте разрешающего элемента получаем 1. В остальных клетках столбца записываем нули.

Таким образом, в новом плане 1 заполнены строка и столбец . Все остальные элементы нового плана 1, включая элементы индексной строки, определяются по правилу прямоугольника.

Для этого выбираем из старого плана четыре числа, которые расположены в вершинах прямоугольника и всегда включают разрешающий элемент РЭ.

СТЭ - элемент старого плана, РЭ - разрешающий элемент (48), А и В - элементы старого плана, образующие прямоугольник с элементами СТЭ и РЭ.

Представим расчет каждого элемента в виде таблицы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Получаем новую симплекс-таблицу:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 |
| x2 |  | 0 | 1 |  | 0 |  |
| x1 |  | 1 | 0 |  | 0 |  |
| x4 |  | 0 | 0 |  | 1 |  |
|  |  | 0 | 0 |  | 0 |  |

**Итерация №1.**

**1. Проверка критерия оптимальности.**

Текущий опорный план неоптимален, так как в индексной строке находятся отрицательные коэффициенты.

**2. Определение новой базисной переменной.**

В качестве ведущего выберем столбец, соответствующий переменной , так как это наибольший коэффициент по модулю.

**3. Определение новой свободной переменной.**

Вычислим значения по строкам как частное от деления и из них выберем наименьшее:

Следовательно, 1-ая строка является ведущей.

Разрешающий элемент равен () и находится на пересечении ведущего столбца и ведущей строки.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 | min |
| x2 |  | 0 | 1 |  | 0 |  |  |
| x1 |  | 1 | 0 |  | 0 |  | - |
| x4 |  | 0 | 0 |  | 1 |  | - |
|  |  | 0 | 0 |  | 0 |  | 0 |

**4. Пересчет симплекс-таблицы.**

Формируем следующую часть симплексной таблицы. Вместо переменной в план 2 войдет переменная .

Строка, соответствующая переменной в плане 2, получена в результате деления всех элементов строки плана 1 на разрешающий элемент . На месте разрешающего элемента получаем 1. В остальных клетках столбца записываем нули.

Таким образом, в новом плане 2 заполнены строка и столбец . Все остальные элементы нового плана 2, включая элементы индексной строки, определяются по правилу прямоугольника.

Представим расчет каждого элемента в виде таблицы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Получаем новую симплекс-таблицу:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 |
| x5 |  | 0 |  |  | 0 | 1 |
| x1 |  | 1 |  |  | 0 | 0 |
| x4 |  | 0 |  |  | 1 | 0 |
|  |  | 0 |  |  | 0 | 0 |

**1. Проверка критерия оптимальности.**

Среди значений индексной строки нет отрицательных. Поэтому эта таблица определяет оптимальный план задачи.

Окончательный вариант симплекс-таблицы:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 |
| x5 |  | 0 |  |  | 0 | 1 |
| x1 |  | 1 |  |  | 0 | 0 |
| x4 |  | 0 |  |  | 1 | 0 |
|  |  | 0 |  |  | 0 | 0 |

Оптимальный план можно записать так:

**Метод Гомори.**

В полученном оптимальном плане присутствуют дробные числа.

По 1-у уравнению с переменной , получившей нецелочисленное значение в оптимальном плане с наибольшей дробной частью , составляем дополнительное ограничение:

Дополнительное ограничение имеет вид:

Преобразуем полученное неравенство в уравнение:

коэффициенты которого введем дополнительной строкой в оптимальную симплексную таблицу.

Поскольку двойственный симплекс-метод используется для поиска минимума целевой функции, делаем преобразование F(x) = -F(X).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 | x6 |
| x5 |  | 0 |  |  | 0 | 1 | 0 |
| x1 |  | 1 |  |  | 0 | 0 | 0 |
| x4 |  | 0 |  |  | 1 | 0 | 0 |
| x6 |  | 0 |  |  | 0 | 0 | 1 |
|  |  | 0 |  |  | 0 | 0 | 0 |

**1. Проверка критерия оптимальности.**

План 0 в симплексной таблице является псевдопланом, поэтому определяем ведущие строку и столбец.

**2. Определение новой свободной переменной.**

Среди отрицательных значений базисных переменных выбираем наибольший по модулю.

Ведущей будет 4-ая строка, а переменную следует вывести из базиса.

**3. Определение новой базисной переменной.**

Минимальное значение θ соответствует 3-му столбцу, т.е. переменную x3 необходимо ввести в базис.

На пересечении ведущих строки и столбца находится разрешающий элемент (РЭ), равный ).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**4. Пересчет симплекс-таблицы.**

Выполняем преобразования симплексной таблицы методом Жордано- Гаусса.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 | x6 |
| x5 | 0 | 0 | 9 | 0 | 0 | 1 | 1 |
| x1 | 9 | 1 | 0 | 0 | 0 | 0 |  |
| x4 | 100 | 0 | 5 | 0 | 1 | 0 | 7 |
| x3 | 3 | 0 | 3 | 1 | 0 | 0 | -5 |
|  | 0 | 0 | -31 | 0 | 0 | 0 |  |

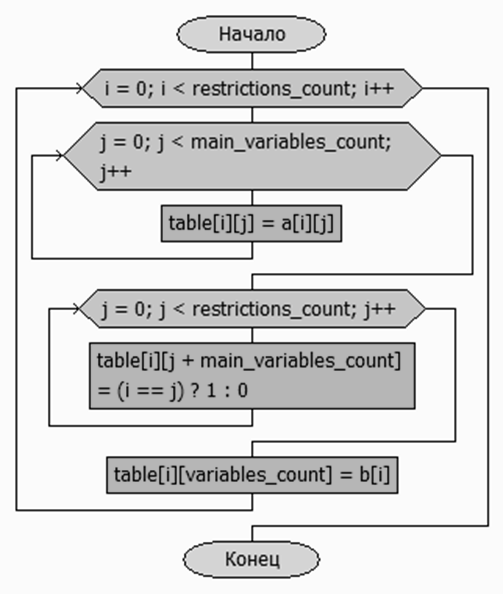
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Решение получилось целочисленным. Нет необходимости применять метод Гомори.

Оптимальный целочисленный план можно записать так:

**Блок – схемы:**

Функция Init\_Table



Функция get\_negative\_b\_row

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Функция get\_negative\_b\_column

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Функция remove\_negative\_b

Изображение выглядит как текст, диаграмма, линия, Параллельный

Автоматически созданное описание

Функция gauss

Изображение выглядит как текст, диаграмма, зарисовка, рисунок

Автоматически созданное описание

Функция calculate\_f

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Функция get\_relations

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

Функция get\_solve

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Функция get\_first\_real

Изображение выглядит как текст, диаграмма, снимок экрана, зарисовка

Автоматически созданное описание

Функция solve

Изображение выглядит как зарисовка, рисунок, диаграмма

Автоматически созданное описание

Функция part

Изображение выглядит как текст, Шрифт, диаграмма, снимок экрана

Автоматически созданное описание

Функция make\_gamori

Изображение выглядит как текст, зарисовка, диаграмма, рисунок

Автоматически созданное описание

Функция solve\_integer

Изображение выглядит как текст, черно-белый, диаграмма, Шрифт

Автоматически созданное описание

Функция print\_best\_solve

Изображение выглядит как текст, диаграмма, зарисовка, рисунок

Автоматически созданное описание

Функция print\_table

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Функция print\_coef

Изображение выглядит как текст, диаграмма, План, зарисовка

Автоматически созданное описание

Функция print\_task

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Функция main

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

**Код программы:**

Алгоритм Гомори

import numpy as np  
from copy import deepcopy  
from itertools import product  
  
MAX\_MODE = 'MAX' # режим максимизации  
MIN\_MODE = 'MIN' # режим минимизации  
  
class SimplexMethod:  
 def \_\_init\_\_(self, c, a, b, mode):  
 self.main\_variables\_count = a.shape[1] # количество переменных  
 self.restrictions\_count = a.shape[0] # количество ограничений  
 self.variables\_count = self.main\_variables\_count + self.restrictions\_count # количество переменных  
 self.mode = mode # запоминаем режим работы  
 self.c = np.concatenate([c, np.zeros((self.restrictions\_count + 1))]) # коэффициенты функции  
 self.f = np.zeros((self.variables\_count + 1)) # значения функции F  
 self.basis = [i + self.main\_variables\_count for i in range(self.restrictions\_count)] # индексы базисных переменных  
 self.task = {'a': deepcopy(a), 'b': deepcopy(b), 'c': deepcopy(c)} # сохраняем начальную задачу  
 self.init\_table(a, b) # инициализация таблицы  
  
 def init\_table(self, a, b):  
 self.table = np.zeros((self.restrictions\_count, self.variables\_count + 1)) # коэффициенты таблицы  
 for i in range(self.restrictions\_count):  
 for j in range(self.main\_variables\_count):  
 self.table[i][j] = a[i][j]  
 for j in range(self.restrictions\_count):  
 self.table[i][j + self.main\_variables\_count] = int(i == j)  
 self.table[i][-1] = b[i]  
  
 # получение строки с максимальным по модулю отрицательным значением b  
 def get\_negative\_b\_row(self):  
 row = -1  
 for i, a\_row in enumerate(self.table):  
 if a\_row[-1] < 0 and (row == -1 or abs(a\_row[-1]) > abs(self.table[row][-1])):  
 row = i  
 return row  
  
 # получение столбца с максимальным по модулю элементом в строке  
 def get\_negative\_b\_column(self, row):  
 column = -1  
 for i, aij in enumerate(self.table[row][:-1]):  
 if aij < 0 and (column == -1 or abs(aij) > abs(self.table[row][column])):  
 column = i  
 return column  
  
 # удаление отрицательных свободных коэффициентов  
 def remove\_negative\_b(self):  
 while True:  
 row = self.get\_negative\_b\_row() # ищем строку, в которой находится отрицательное b  
 if row == -1: # если не нашли такую строку  
 return True # то всё хорошо  
 column = self.get\_negative\_b\_column(row) # ищем разрешающий столбец  
 if column == -1:  
 return False # не удалось удалить  
 self.gauss(row, column) # выполняем исключение гаусса  
 self.calculate\_f()  
  
 # выполнение шага метода гаусса  
 def gauss(self, row, column):  
 self.table[row] /= self.table[row][column]  
 for i in range(self.restrictions\_count):  
 if i != row:  
 self.table[i] -= self.table[row] \* self.table[i][column]  
 for j in range(self.variables\_count):  
 if abs(self.table[i][j]) < 1e-10:  
 self.table[i][j] = 0  
 self.basis[row] = column # делаем переменную базисной  
  
 # расчёт значений F  
 def calculate\_f(self):  
 for i in range(self.variables\_count + 1):  
 self.f[i] = -self.c[i]  
 for j in range(self.restrictions\_count):  
 self.f[i] += self.c[self.basis[j]] \* self.table[j][i]  
  
 # расчёт симплекс-отношений для столбца column  
 def get\_relations(self, column):  
 q = []  
 for i in range(self.restrictions\_count):  
 if self.table[i][column] == 0:  
 q.append(np.inf)  
 else:  
 q\_i = self.table[i][-1] / self.table[i][column]  
 q.append(q\_i if q\_i >= 0 else np.inf)  
 return q  
  
 # получение решения  
 def get\_solve(self):  
 x = np.zeros((self.variables\_count))  
 # заполняем решение  
 for i in range(self.restrictions\_count):  
 x[self.basis[i]] = self.table[i][-1]  
 return x # возвращаем полученное решение  
  
 # получение первого вещественного решения  
 def get\_first\_real(self, x):  
 return next((i for i, xi in enumerate(x[:self.main\_variables\_count]) if xi != int(xi)), -1)  
  
 # решение  
 def solve(self, debug):  
 self.calculate\_f()  
 if debug:  
 print('\nIteration 0')  
 self.print\_table()  
 if not self.remove\_negative\_b():  
 print('Solve does not exist')  
 return False  
 iteration = 1  
 while True:  
 self.calculate\_f()  
 if debug:  
 print('\nIteration', iteration)  
 self.print\_table()  
 if all(fi >= 0 if self.mode == MAX\_MODE else fi <= 0 for fi in self.f[:-1]): # если план оптимален  
 break # то завершаем работу  
 column = (np.argmin if self.mode == MAX\_MODE else np.argmax)(self.f[:-1]) # получаем разрешающий столбец  
 q = self.get\_relations(column) # получаем симплекс-отношения для найденного столбца  
 if all(qi == np.inf for qi in q): # если не удалось найти разрешающую строку  
 print('Solve does not exist') # сообщаем, что решения нет  
 return False  
 self.gauss(np.argmin(q), column) # выполняем исключение гаусса  
 iteration += 1  
 return True # решение есть  
  
 # получение дробной части  
 def part(self, a):  
 if a > 0:  
 return a - int(a)  
 return int(abs(a)) + 1 + a  
  
 def make\_gamori(self, x, real\_index, debug):  
 b = self.part(x[real\_index])  
 r\_main = [0 if i in self.basis else self.part(self.table[real\_index][i]) for i in range(self.main\_variables\_count)]  
 r\_basis = [0 if i in self.basis else self.part(self.table[real\_index][i]) for i in range(self.main\_variables\_count, self.variables\_count)]  
 # добавляем условия  
 task1\_a = np.append(self.task['a'], np.array([r\_main]), 0)  
 task1\_b = np.append(self.task['b'], -b)  
 simplex1 = SimplexMethod(self.task['c'], task1\_a, task1\_b, self.mode)  
 for i in range(self.restrictions\_count):  
 for j in range(self.variables\_count):  
 simplex1.table[i][j] = self.table[i][j]  
 simplex1.table[i][-1] = self.table[i][-1]  
 for i in range(self.main\_variables\_count):  
 simplex1.table[-1][i] = -r\_main[i]  
 for i in range(self.main\_variables\_count, self.variables\_count):  
 simplex1.table[-1][i] = -r\_basis[i - self.main\_variables\_count]  
 return simplex1.solve\_integer(debug)  
  
 # получение целочисленных решений  
 def solve\_integer(self, debug=False):  
 print('Start solving task:')  
 self.print\_task()  
 # если решение не было найдено, то добавляем пустое решение  
 if not self.solve(debug):  
 return []  
 print("Solve:", self.get\_solve()[:self.main\_variables\_count], ', F:', self.f[-1], '\n')  
 x = self.get\_solve()  
 real\_index = self.get\_first\_real(x)  
 # если решение содержало только целые числа, то возвращаем решение  
 if real\_index == -1:  
 return [(x, self.f[-1])]  
 return self.make\_gamori(x, real\_index, debug)  
  
 # вывод лучшего решения  
 def print\_best\_solve(self, solves):  
 sign = 1 if self.mode == MAX\_MODE else -1  
 imax = 0  
 for i, solve in enumerate(solves):  
 if solve[1] \* sign > solves[imax][1] \* sign:  
 imax = i  
 x, f = solves[imax]  
 print('\nBest solve: ', x[:self.main\_variables\_count], 'F:', f)  
  
 # вывод симплекс-таблицы  
 def print\_table(self):  
 print(' |' + ''.join([' x%-3d |' % (i + 1) for i in range(self.variables\_count)]) + ' b |')  
 for i in range(self.restrictions\_count):  
 print('%4s |' % ('x' + str(self.basis[i] + 1)) + ''.join([' %6.2f |' % aij for j, aij in enumerate(self.table[i])]))  
 print(' F |' + ''.join([' %6.2f |' %  
  
 aij for aij in self.f]))  
 # print(' x |' + ''.join([' %6.2f |' % xi for xi in self.get\_solve()]))  
  
 # вывод коэффициента  
 def print\_coef(self, ai, i):  
 if ai == 1:  
 return 'x%d' % (i + 1)  
 if ai == -1:  
 return '-x%d' % (i + 1)  
 return '%.2fx%d' % (ai, i + 1)  
  
 # вывод задачи  
 def print\_task(self, full=True):  
 print(' + '.join(['%.2fx%d' % (ci, i + 1) for i, ci in enumerate(self.c[:self.main\_variables\_count]) if ci != 0]), '-> ', self.mode)  
 for row in self.table:  
 if full:  
 print(' + '.join([self.print\_coef(ai, i) for i, ai in enumerate(row[:self.variables\_count]) if ai != 0]), '=', row[-1])  
 else:  
 print(' + '.join([self.print\_coef(ai, i) for i, ai in enumerate(row[:self.main\_variables\_count]) if ai != 0]), '<=', row[-1])  
  
def main():  
 c = np.array([9, -4, 0, 0, 3])  
 a = np.array([  
 [10, 3, 1, 0, 0],  
 [14, -5, 0, -1, 0],  
 [2, -9, 0, 0, -1]  
 ])  
 b = np.array([93, 26, 18])  
 simplex = SimplexMethod(c, a, b, MAX\_MODE)  
 print("Solve with Gomory method:")  
 solves = simplex.solve\_integer(True)  
 simplex.print\_best\_solve(solves)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Алгоритм ветвей и границ

import heapq  
  
  
def branch\_and\_bound(graph, start\_node):  
 n = len(graph)  
 min\_heap = []  
 for i in range(1, n):  
 heapq.heappush(min\_heap, (graph[start\_node][i], [start\_node, i]))  
  
 while min\_heap:  
 current\_cost, current\_path = heapq.heappop(min\_heap)  
 last\_node = current\_path[-1]  
  
 if len(current\_path) == n:  
 if graph[last\_node][start\_node] != 0:  
 return current\_cost + graph[last\_node][start\_node], current\_path  
 else:  
 for i in range(n):  
 if i not in current\_path and graph[last\_node][i] != 0:  
 heapq.heappush(min\_heap, (current\_cost + graph[last\_node][i], current\_path + [i]))

Результат работы программы:

Solve with Gomory method:  
Start solving task:  
9.00x1 + -4.00x2 + 3.00x5 -> MAX  
10.00x1 + 3.00x2 + x3 + x6 = 93.0  
14.00x1 + -5.00x2 + -x4 + x7 = 26.0  
2.00x1 + -9.00x2 + -x5 + x8 = 18.0  
  
Iteration 0  
| x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | b |  
x6 | 10.00 | 3.00 | 1.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 93.00 |  
x7 | 14.00 | -5.00 | 0.00 | -1.00 | 0.00 | 0.00 | 1.00 | 0.00 | 26.00 |  
x8 | 2.00 | -9.00 | 0.00 | 0.00 | -1.00 | 0.00 | 0.00 | 1.00 | 18.00 |  
F | -9.00 | 4.00 | 0.00 | 0.00 | -3.00 | 0.00 | 0.00 | 0.00 | 0.00 |  
  
Iteration 1  
| x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | b |  
x6 | 10.00 | 3.00 | 1.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 93.00 |  
x7 | 14.00 | -5.00 | 0.00 | -1.00 | 0.00 | 0.00 | 1.00 | 0.00 | 26.00 |  
x8 | 2.00 | -9.00 | 0.00 | 0.00 | -1.00 | 0.00 | 0.00 | 1.00 | 18.00 |  
F | -9.00 | 4.00 | 0.00 | 0.00 | -3.00 | 0.00 | 0.00 | 0.00 | 0.00 |  
  
Iteration 2  
| x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | b |  
x6 | 0.00 | 6.57 | 1.00 | 0.71 | 0.00 | 1.00 | -0.71 | 0.00 | 74.43 |  
x1 | 1.00 | -0.36 | 0.00 | -0.07 | 0.00 | 0.00 | 0.07 | 0.00 | 1.86 |  
x8 | 0.00 | -8.29 | 0.00 | 0.14 | -1.00 | 0.00 | -0.14 | 1.00 | 14.29 |  
F | 0.00 | 0.79 | 0.00 | -0.64 | -3.00 | 0.00 | 0.64 | 0.00 | 16.71 |  
  
Optimal plan: x1 = 9, x2 = 0, x3 = 3, x4 = 100, x5 = 0  
Target function: 81

**Вывод**: Освоение методов отсечения Гомори для полностью целочисленных задач является важным аспектом в решении комбинаторных оптимизационных проблем. Метод отсечения Гомори используется для улучшения решений задач целочисленного программирования путем добавления дополнительных целочисленных ограничений, что повышает качество найденного решения. Этот метод основан на выявлении и добавлении так называемых "гомори-неравенств" к существующим ограничениям, что улучшает эффективность поиска оптимального целочисленного решения. Реализация этого алгоритма в программе позволяет проводить более глубокий и эффективный поиск оптимального решения в целочисленном программировании, что улучшает результаты решения различных оптимизационных задач.