**Министерство образования Российской Федерации**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**им. Н.Э. БАУМАНА**

Факультет: Информатика и системы управления Кафедра: Информационная безопасность (ИУ8)

**Методы оптимизации Лабораторная работа №3 на тему:**

«Целочисленное линейное программирование»

Вариант 18

**Преподаватель:**

Коннова Н.С.

**Студент**:  
Ожогин М.А.

**Группа:**

ИУ8-34

Москва 2024

# Цель работы

Изучить постановку задачи целочисленного линейного программирования; получить решение задачи ЦЛП методом ветвей и границ.

# Постановка задачи

Условие ЦЛП: Требуется выбрать среди всех n-мерных векторов x = (x1, x2, …, xn) (xi **∊** Z+0, xi **≥** 0 для I = 1, …, n), удовлетворяющих системе

такой для которого достигается минимум ЦФ:

Задача ЦЛП у нас будет иметь вид:

Здесь x = [x1, x2, …, xn]T – вектор решения;

c = [c1, c2, …, cn] – вектор коэффициентов ЦФ F;

A = – матрица системы ограничений;

b = [b1, b2, …, bm]T – вектор правой части системы ограничений.

Требуется решить задачу вида методом ветвей и границ согласно описанному алгоритму, привести все шаги, построить дерево решения, найти оптимальный план; решить эту задачу также методом полного перебора, привести все допустимые решения и значения ЦФ для них; сравнить полученные по обоим методам результаты.

# Ход работы

Рассмотрим задачу ЦЛП

Решим исходную задачу ЛП с помощью симплекс метода:

| b | x1 | x2 | x3

-------------------------------------

x4 | 8.00 | 2.00 | 1.00 | 1.00

x5 | 2.00 | 1.00 | 2.00 | 0.00

x6 | 6.00 | 0.00 | 0.50 | 4.00

F | 0.00 | 7.00 | 7.00 | 6.00

Resolving element located: [1, 1, 0]

| b | x5 | x2 | x3

-----------------------------------------------

x4 | 4.00 | -2.00 | -3.00 | 1.00

x1 | 2.00 | 1.00 | 2.00 | 0.00

x6 | 6.00 | -0.00 | 0.50 | 4.00

F | -14.00 | -7.00 | -7.00 | 6.00

Resolving element located: [4.0, 2, 2]

Optimal Solution Found:

| b | x5 | x2 | x6

----------------------------------------------------

x4 | 2.50 | -2.00 | -3.12 | -0.25

x1 | 2.00 | 1.00 | 2.00 | -0.00

x3 | 1.50 | -0.00 | 0.12 | 0.25

F | -23.00 | -7.00 | -7.75 | -1.50

Решение задачи ЛП симплекс-методом:

Осуществим ветвление по переменной x3:

1. Введем новое ограничение x3 **≥** 2 и получим задачу ЦЛП:

Решим составленное уравнение с помощью симплекс-метода.

| b | x1 | x2 | x3

------------------------------------------

x4 | 8.00 | 2.00 | 1.00 | 1.00

x5 | 2.00 | 1.00 | 2.00 | 0.00

x6 | 6.00 | 0.00 | 0.50 | 4.00

x7 | -2.00 | 0.00 | 0.00 | -1.00

F | 0.00 | 7.00 | 7.00 | 6.00

Resolving element located: [4, 2, 2]

| b | x1 | x2 | x6

-----------------------------------------------

x4 | 6.50 | 2.00 | 0.88 | -0.25

x5 | 2.00 | 1.00 | 2.00 | -0.00

x3 | 1.50 | 0.00 | 0.12 | 0.25

x7 | -0.50 | 0.00 | 0.12 | 0.25

F | -9.00 | 7.00 | 6.25 | -1.50

Ветвь не имеет решения

1. Введем новое ограничение x3 **≤** 1 и получим задачу ЦЛП:

| b | x1 | x2 | x3

-------------------------------------

x4 | 8.00 | 2.00 | 1.00 | 1.00

x5 | 2.00 | 1.00 | 2.00 | 0.00

x6 | 6.00 | 0.00 | 0.50 | 4.00

x7 | 1.00 | 0.00 | 0.00 | 1.00

F | 0.00 | 7.00 | 7.00 | 6.00

Resolving element located: [1, 1, 0]

| b | x5 | x2 | x3

-----------------------------------------------

x4 | 4.00 | -2.00 | -3.00 | 1.00

x1 | 2.00 | 1.00 | 2.00 | 0.00

x6 | 6.00 | -0.00 | 0.50 | 4.00

x7 | 1.00 | -0.00 | 0.00 | 1.00

F | -14.00 | -7.00 | -7.00 | 6.00

Resolving element located: [1.0, 3, 2]

Optimal Solution Found:

| b | x5 | x2 | x7

-----------------------------------------------

x4 | 3.00 | -2.00 | -3.00 | -1.00

x1 | 2.00 | 1.00 | 2.00 | -0.00

x6 | 2.00 | 0.00 | 0.50 | -4.00

x3 | 1.00 | -0.00 | 0.00 | 1.00

F | -20.00 | -7.00 | -7.00 | -6.00

The function is maximized.

New best solution found: [20.0, 2.0, 0, 1.0]

Составим дерево решений по методу ветвей и границ:

F = 23  
x1=1 x2=0 x3=1.5

Решения не существует

F = 20  
x1=2 x2=0 x3=1

Далее решим эту же задачу методом полного перебора

For c: [7 7 6]

[0 0 0] = 0

[0 0 1] = 6

[0 1 0] = 7

[0 1 1] = 13

[1 0 0] = 7

[1 0 1] = 13

[2 0 0] = 14

[2 0 1] = 20

Brute-force: [20, [2, 0, 1]]

Методом полного перебора мы получили ответ такой же, как и при использовании симплекс метода для поиска целочисленного решения.

Итоговый ответ:

F = 20; x1 = 2; x2 = 0; x3 = 1

# Вывод

В ходе выполнения лабораторной работы были изучены основные принципы целочисленного программирования, был изучен метод ветвей и границ, построено дерево, а также метод полного перебора с проверкой получившихся значений

# Приложение.

Файл “main.py”:

import branches  
import bruteforce  
def main():  
 c = [7, 7, 6]  
 A = [[2, 1, 1],  
 [1, 2, 0],  
 [0, 0.5, 4]]  
 b = [8, 2, 6]  
 f = 0  
 minimize = False  
  
 print("Best answer: ", branches.method\_execute(c, A, b, f, minimize))  
 print("Brute-force: ", bruteforce.method\_execute(c, A, b, f, minimize))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Файл “bruteforce.py”:

import numpy as np  
from itertools import product  
  
def method\_execute(c, A, b, f, minimize):  
 c = np.array(c)  
 A = np.array(A)  
 b = np.array(b)  
  
 upper\_bounds = np.ceil(b / np.min(A + (A == 0), axis=0)).astype(int)  
 x\_ranges = [range(0, ub + 1) for ub in upper\_bounds]  
  
 max\_F = float('-inf')  
 best\_x = None  
  
 print("For c:", c)  
 for x in product(\*x\_ranges):  
 x = np.array(x)  
 if np.all(np.dot(A, x) <= b):  
 F = np.dot(c, x)  
 print(x, " = ", F)  
 if F > max\_F:  
 max\_F = F  
 best\_x = x  
  
 return [int(F), best\_x.tolist()]

Файл “branches.py”

from simp\_solv import execute\_simplex  
from math import floor  
  
  
def method\_execute(c, A, b, f, minimize, best\_solution=None):  
 stack = [(c, A, b, f, minimize, best\_solution)]  
  
 while stack:  
 current\_c, current\_A, current\_b, current\_f, current\_minimize, current\_best\_solution = stack.pop()  
 answer\_simplexsus = execute\_simplex(current\_c, current\_A, current\_b, current\_f, current\_minimize)  
 if answer\_simplexsus[0] == float("inf"):  
 print("Ветвь не имеет решения")  
 continue  
 answer\_variables = answer\_simplexsus[1::]  
 current\_best\_solution = check\_integer\_solution(answer\_simplexsus, answer\_variables, current\_best\_solution)  
 if current\_best\_solution is not None and (best\_solution is None or current\_best\_solution[0] > best\_solution[0]):  
 best\_solution = current\_best\_solution  
 i = 0  
 found = False  
 while i < len(answer\_variables) and not found:  
 if not is\_integer(answer\_variables[i]):  
 branching\_variable = floor(answer\_variables[i])  
 print(  
 f"Adding a new condition for x{i + 1} = {answer\_variables[i]}:",  
 f"[x{i + 1} <= {branching\_variable}; x{i + 1} >= {branching\_variable + 1}]",  
 )  
  
 # Создаем новые ограничения  
 new\_A\_left = current\_A + [[0 if j != i else 1 for j in range(len(c))]]  
 new\_b\_left = current\_b + [branching\_variable]  
  
 # Добавляем новое ограничение для x\_i <= branching\_variable  
 stack.append((current\_c, new\_A\_left, new\_b\_left, current\_f, current\_minimize, best\_solution))  
  
 # Добавляем ограничение для x\_i >= branching\_variable + 1  
 new\_A\_right = current\_A + [[0 if j != i else -1 for j in range(len(c))]]  
 new\_b\_right = current\_b + [(branching\_variable + 1) \* -1]  
  
 stack.append((current\_c, new\_A\_right, new\_b\_right, current\_f, current\_minimize, best\_solution))  
  
 found = True  
  
 i += 1  
  
 return best\_solution  
  
  
def check\_integer\_solution(answer\_simplexsus, answer\_variables, best\_solution):  
 if all(is\_integer(var) for var in answer\_variables):  
 if best\_solution is None or answer\_simplexsus[0] < best\_solution[0]:  
 best\_solution = answer\_simplexsus  
 print("New best solution found:", best\_solution, "\n")  
 else:  
 print("Current solution is integer but not better:", answer\_simplexsus)  
  
 return best\_solution  
  
  
def is\_integer(value):  
 return floor(value) == value

Файл “simp\_solv.py”

columns = []  
rows = []  
old\_columns = []  
  
def init\_headers(c, A, b):  
 num\_columns = len(c)  
 num\_rows = len(b)  
 global columns  
 global rows  
 global old\_columns  
 columns = ['b'] + [f'x{i + 1}' for i in range(num\_columns)]  
 old\_columns = columns.copy()  
 rows = []  
 for i in range(num\_rows):  
 temp\_row = [f'x{i + num\_columns + 1}']  
 rows.append(temp\_row)  
 rows.append('F')  
  
def validate\_simplex\_input(c, A, b):  
 len\_A = len(A[0])  
 return (all(len(row) == len\_A for row in A) and  
 len(c) == len\_A and  
 len(b) == len(A))  
  
def check\_solution\_existence(c, A, b):  
 if all(x == 0 for x in c):  
 return False  
 return all(b[row] >= 0 or min(A[row]) < 0 for row in range(len(b)))  
  
def construct\_simplex\_table(c, A, b, f):  
 table = []  
 for i in range(len(A)):  
 table.append([b[i]] + A[i])  
 table.append([f] + c)  
 return table  
  
def format\_header\_value(value):  
 return " ".join(str(v) for v in value) if isinstance(value, list) else str(value)  
  
def display\_simplex\_table(simplex\_table):  
 print()  
 formatted\_columns = [format\_header\_value(col) for col in columns]  
 formatted\_rows = [format\_header\_value(row) for row in rows]  
 max\_width = max(len(str(float(j))) for row in simplex\_table for j in row if isinstance(j, (int, float))) + 2  
 full\_headers = [""] + formatted\_columns # Добавляем 'b' перед заголовками столбцов  
 print(" | ".join(f"{header:>{max\_width}}" for header in full\_headers))  
 print("-" \* (max\_width \* len(full\_headers) + 3 \* (len(full\_headers) - 1)))  
 for i, row in enumerate(simplex\_table):  
 row\_values = [formatted\_rows[i]] + list(row) # Добавляем заголовок для строки 'b'  
 print(" | ".join(  
 f"{float(j):>{max\_width}.2f}" if isinstance(j, (int, float)) else str(j).ljust(max\_width) for j in  
 row\_values))  
  
def locate\_resolving\_element(c, A, b):  
 if not check\_solution\_existence(c, A, b):  
 return ["not\_er"]  
 for row in range(len(b)):  
 if b[row] < 0:  
 for col in range(len(A[0])):  
 if A[row][col] < 0:  
 try:  
 return calculate\_min\_ratio(A, b, col)  
 except:  
 return ["inf\_er"]  
 if max(c) < 0:  
 return ["not\_er"]  
 c\_max\_index = c.index(max(c))  
 return calculate\_min\_ratio(A, b, c\_max\_index)  
  
def calculate\_min\_ratio(A, b, ratio\_col):  
 min\_ratio = float("inf")  
 min\_ratio\_row = -1  
 for row in range(len(A)):  
 if A[row][ratio\_col] == 0:  
 continue  
 ratio = b[row] / A[row][ratio\_col]  
 if 0 < ratio < min\_ratio:  
 min\_ratio = ratio  
 min\_ratio\_row = row  
 if min\_ratio\_row == -1:  
 raise ValueError("No valid resolving element found.")  
 return [A[min\_ratio\_row][ratio\_col], min\_ratio\_row, ratio\_col]  
  
def perform\_simplex\_iteration(c, A, b, f, res\_el):  
 global columns, rows  
 new\_resolving\_element = 1 / res\_el[0]  
 new\_b = [0] \* len(b)  
 new\_A = [[0] \* len(A[0]) for \_ in A]  
 for i in range(len(A)):  
 new\_A[i][res\_el[2]] = (  
 new\_resolving\_element if i == res\_el[1]  
 else A[i][res\_el[2]] / res\_el[0] \* -1  
 )  
  
 new\_c = [0] \* len(c)  
 new\_c[res\_el[2]] = c[res\_el[2]] / res\_el[0] \* -1  
 for i in range(len(A[0])):  
 if i != res\_el[2]:  
 new\_A[res\_el[1]][i] = A[res\_el[1]][i] / res\_el[0]  
  
 new\_b[res\_el[1]] = b[res\_el[1]] / res\_el[0]  
 for i in range(len(c)):  
 if i == res\_el[2]:  
 continue  
 new\_c[i] = c[i] - (A[res\_el[1]][i] \* c[res\_el[2]]) / (res\_el[0])  
  
 for i in range(len(b)):  
 if i == res\_el[1]:  
 continue  
 new\_b[i] = b[i] - ((A[i][res\_el[2]] \* b[res\_el[1]]) / res\_el[0])  
  
 for i in range(len(A)):  
 for j in range(len(A[0])):  
 if (i == res\_el[1]) or (j == res\_el[2]):  
 continue  
 new\_A[i][j] = A[i][j] - ((A[i][res\_el[2]] \* A[res\_el[1]][j]) / res\_el[0])  
  
 new\_f = f - ((c[res\_el[2]] \* b[res\_el[1]]) / res\_el[0])  
 temp\_row = rows[res\_el[1]]  
 rows[res\_el[1]] = columns[1+res\_el[2]]  
 columns[1+res\_el[2]] = temp\_row  
 return new\_c, new\_A, new\_b, new\_f  
  
def execute\_simplex(c, A, b, f, minimize):  
 if validate\_simplex\_input(c, A, b):  
 print("Input Validation: Passed")  
 init\_headers(c, A, b)  
 if minimize:  
 c = [-x for x in c]  
 while (max(c) > 0) or (min(b) < 0):  
 simplex\_table = construct\_simplex\_table(c, A, b, f)  
 display\_simplex\_table(simplex\_table)  
 resolving\_element = locate\_resolving\_element(c, A, b)  
 if resolving\_element == ["not\_er"]:  
 print("No feasible solution exists.")  
 return [float("inf")]  
 if resolving\_element == ["inf\_er"]:  
 print("Infinite solutions detected.")  
 return [float("inf")]  
 print("Resolving element located:", resolving\_element)  
 c, A, b, f = perform\_simplex\_iteration(c, A, b, f, resolving\_element)  
  
 print("\nOptimal Solution Found:")  
 simplex\_table = construct\_simplex\_table(c, A, b, f)  
 display\_simplex\_table(simplex\_table)  
 else:  
 print("Input Validation: Failed")  
 return [float("inf")]  
  
 if minimize:  
 print("The function is minimized.")  
 return [f] + create\_answer\_variables(b, rows, old\_columns)  
 print("The function is maximized.")  
 return [f \* -1] + create\_answer\_variables(b, rows, old\_columns)  
  
def create\_answer\_variables(b, var\_row, old\_var\_col):  
 answer\_variables = [0 for \_ in range(len(old\_var\_col) - 1)]  
 for i in range(len(var\_row)):  
 if var\_row[i] in old\_var\_col:  
 answer\_variables[old\_var\_col.index(var\_row[i]) - 1] = round(b[i], 2) # Сохранение значения  
 return answer\_variables