Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа №1

«Метод ветвей и границ»

Выполнил: ст. гр. 953503

Басенко К. А.

Проверил: Дугинов О. И.

Минск 2022

**Постановка задачи**

Пусть имеется задача целочисленного линейного программирования с двусторонними ограничениями

(1)

,

где ∈ Zn, = (, , . . . , )T ∈ Zn — вектор переменных, A ∈ Zm×n —целочисленная матрица, в которой m строк и n столбцов, ∈ Zm, ∈ Zn и ∈ Zn. Требуется определить совместна ли задача и в случае положительного ответа найти оптимальный план. Это можно сделать с помощью метода ветвей и границ. Цель настоящей лабораторной работы — реализовать метод ветвей и границ для решения задач целочисленного линейного программирования с

двусторонними ограничениями.

**Описание алгоритма метода**

Шаг 1. Преобразуем задачу (1) таким образом, чтобы ⩽ 0. Для каждого индекса i ∈ {1, 2, . . . , n} такого, что i > 0

а) в векторе i-ую компоненту умножим на −1;

б) в матрице i-ый столбец умножим на −1;

в) в каждом из векторов и i-ую компоненту умножим на −1;

г) i-ую компоненту вектора и i-ую компоненту вектора поменяем местами.

Шаг 2. Отбросим условие целочисленности на переменные и приведем полученную задачу линейного программирования в каноническую форму без учета ограничений неотрицательности

(2)

,

где

α = 0,

x = (x1, x2, . . . , xn, xn+1, xn+2, . . . , x2n+m)T ∈ R2n+m,

c = (c1, c2, . . . , cn, 0, 0, . . . , 0)⊺ ∈ Z2n+m

и

, , .

Шаг 3. Создадим переменные x\*, r и пустой стек S. В переменной x\* будем хранить наилучший допустимый целочисленный план задачи (2). Значение целевой функции задачи (2) на плане x\* будем хранить в переменной r, т.е. r = cTx\*. Поместим в стек S задачу (2) вместе с вектором ∆ = d−.

Шаг 4. Рассмотрим два случая в зависимости от того пустой стек S или нет.

Случай 1. Пусть стек S пустой. Метод завершает свою работу. Если в переменной x\* нет плана, то возвращаем сообщение «задача (1) несовместна». Иначе x\* — это оптимальный план задачи (2) и, в этом случае, возвратим оптимальный план x задачи (1), восстановленный по x\*, следующим образом: для каждого индекса i ∈ {1, 2, . . . , n}

относительно вектора c стоимостей исходной задачи (1).

Случай 2. Пусть стек S непустой. Извлечем из стека S задачу линейного программирования

,

с вектором ∆ ∈ Z2n+m. Заметим, что вектор d− не всегда нулевой. Приведем эту задачу к канонической форме

(3)

,

где a’ = a + cTd−, b’=b−Ad−. Построим начальный базисный двойственный план (y, B), в котором y = (0, 0, …, 0) ∈ Rm+n и B = {j1 = n + 1, j2 = n + 2, . . . , jn+m = 2n + m}. Решим задачу (3) двойственным симплекс-методом и найдем оптимальный план .

Рассмотрим два случая в зависимости от того является план целочисленным или нет.

Случай 1. Пусть план целочисленный. По этому плану восстановим допустимый план задачи (2) следующим образом. Прибавим к плану вектор ∆. Полученный в результате вектор обозначим через xˆ. Если в переменной x\* нет плана или r < cTxˆ+α′ , то запишем в переменную x\* план xˆ, а в переменную r значение cTxˆ + α′.

Случай 2. Пусть план дробный. Выберем дробную компоненту i из числа первых n компонент плана . Если в переменной x\* нет плана или ⌊cT + α′⌋ > r, то построим две новые задачи линейного программирования

,

где вектор b’’ получается из вектора b’ заменой (m+i)-ой компоненты на ⌊i⌋ и

'

,

где d− — это (2n+m)-мерный вектор, который получается из нулевого вектора заменой i-ой компоненты на ⌈i⌉. Обе задачи поместим в стек S вместе с вектором ∆ + d−.

Повторим шаг 4.

**Результат работы**

**Тест 1**

Задание:

Вывод программы:



**Тест 2**

Задание:

Вывод программы:



**Код**

from numbers import Integral

import math

import copy

from math import inf

def dual\_simplex\_max(A, Jb, c, a, b):

while not all(i >= 0 for i in b):

index\_min\_row = min(range(len(b)), key=b.\_\_getitem\_\_)

c = [-i for i in c]

min\_v = inf

index\_min\_column = -1

for i in range(len(A[0])):

if A[index\_min\_row][i] < 0 and abs(c[i] / A[index\_min\_row][i]) < min\_v:

min\_v = abs(c[i] / A[index\_min\_row][i])

index\_min\_column = i

if index\_min\_column == -1:

return None

min\_v = A[index\_min\_row][index\_min\_column]

for i in range(len(A[0])):

A[index\_min\_row][i] = A[index\_min\_row][i] / min\_v

b[index\_min\_row] = b[index\_min\_row] / min\_v

for i in range(len(A)):

if i == index\_min\_row:

continue

value = -A[i][index\_min\_column]

b[i] = b[index\_min\_row] \* value + b[i]

for j in range(len(A[0])):

A[i][j] = A[index\_min\_row][j] \* value + A[i][j]

value = -c[index\_min\_column]

for i in range(len(c)):

c[i] = A[index\_min\_row][i] \* value + c[i]

Jb[index\_min\_row] = index\_min\_column

x = [0 for \_ in range(len(A[0]))]

for i, j in zip(Jb, b):

x[i] = j

return x

def get\_float\_xi(x, n):

for i in range(n):

if not isinstance(x[i], Integral):

return x[i], i

def branch\_and\_bound\_method(i\_c, i\_A, i\_b, i\_d\_minus, i\_d\_plus):

u\_c = copy.deepcopy(i\_c)

u\_A = copy.deepcopy(i\_A)

u\_b = copy.deepcopy(i\_b)

u\_d\_minus = copy.deepcopy(i\_d\_minus)

u\_d\_plus = copy.deepcopy(i\_d\_plus)

n = len(u\_c)

m = len(u\_b)

for i in range(0, n):

if u\_c[i] > 0:

u\_c[i] = u\_c[i] \* (-1)

for j in range(0, m):

u\_A[j][i] = u\_A[j][i] \* (-1)

u\_d\_plus[i] = u\_d\_plus[i] \* (-1)

u\_d\_minus[i] = u\_d\_minus[i] \* (-1)

u\_d\_plus[i], u\_d\_minus[i] = u\_d\_minus[i], u\_d\_plus[i]

a = 0

c = [0 for \_ in range(2 \* n + m)]

for i in range(n):

c[i] = u\_c[i]

A = [[0 for \_ in range(2 \* n + m)] for \_ in range(m + n)]

for i in range(m):

for j in range(n):

A[i][j] = u\_A[i][j]

for i in range(n):

for j in range(n):

if i == j:

A[i + m][j] = 1

for i in range(m + n):

for j in range(m + n):

if i == j:

A[i][j + n] = 1

b = [0 for \_ in range(n + m)]

for i in range(m):

b[i] = u\_b[i]

for i in range(m, n + m):

b[i] = u\_d\_plus[i - m]

d\_minus = [0 for \_ in range(2 \* n + m)]

for i in range(n):

d\_minus[i] = u\_d\_minus[i]

s = []

X = None

r = None

s.append(Stack(c, a, A, b, d\_minus, d\_minus))

while len(s) != 0:

i = len(s) - 1

c = s[i].c

a = s[i].alfa

A = s[i].A

b = s[i].b

d\_minus = s[i].d\_minus

delta = s[i].delta

s.remove(s[i])

if not all(d == 0 for d in d\_minus):

result = 0

for i1, i2 in zip(c, d\_minus):

result += i1 \* i2

a2 = a + result

result = [0 for \_ in range(len(A))]

for i in range(len(A)):

for j in range(len(A[0])):

result[i] += A[i][j] \* d\_minus[j]

b2 = [i1 - i2 for i1, i2 in zip(b, result)]

else:

a2 = a

b2 = b

y = [0 for \_ in range(m + n)]

B = [n + i for i in range(1, n + m + 1)]

Jb = [i - 1 for i in B]

x = dual\_simplex\_max(copy.deepcopy(A), Jb.copy(),

c.copy(), a2, b2.copy())

if all(isinstance(i, Integral) for i in x):

x\_plan = [i + j for i, j in zip(x, delta)]

k = sum([i\*j for i, j in zip(c, x\_plan)]) + a2

if X is None or k > r:

X = x\_plan

r = k

else:

xi, xi\_index = get\_float\_xi(x, n)

k = math.floor(sum([i\*j for i, j in zip(c, x)]) + a2)

if X is None or k > r:

b3 = b2.copy()

b3[m + xi\_index] = math.floor(xi)

d\_minus\_zero = [0 for \_ in d\_minus]

d\_minus = [0 for \_ in range(2 \* n + m)]

d\_minus[xi\_index] = math.ceil(xi)

s.append(Stack(c, a2, A, b3, d\_minus\_zero, delta))

delta = [i + j for i, j in zip(delta, d\_minus)]

s.append(Stack(c, a2, A, b2, d\_minus, delta))

if X is None:

print("Задача несовместна")

return None

else:

x = [0 for \_ in range(n)]

for i in range(n):

if i\_c[i] < 0:

x[i] = X[i]

else:

x[i] = -X[i]

return x

class Stack:

def \_\_init\_\_(self, c, alfa, A, b, d\_minus, delta) -> None:

self.c = c

self.alfa = alfa

self.A = A

self.b = b

self.d\_minus = d\_minus

self.delta = delta

if \_\_name\_\_ == "\_\_main\_\_":

c = [1, 1]

A = [

[5, 9],

[9, 5]

]

b = [63, 63]

d\_minus = [1, 1]

d\_plus = [6, 6]

result = branch\_and\_bound\_method(c, A, b, d\_minus, d\_plus)

if result == None:

print("Задача несовместна")

else:

print(result)