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

$$\begin{aligned} &\bar{c}^T\bar{x} -> \max \\ &\bar{A}\bar{x} \le \bar{b} \\ &\bar{d}^- \le \bar{x} \le \bar{d}^+ , \end{aligned} \qquad (1)$$

где $\bar{c} \in Z^n$, $\bar{x} = (\overline{x1}, \overline{x2}, \ldots, \overline{xn})^T \in Z^n$ — вектор переменных, $A \in Z^{m\times n}$ —целочисленная матрица, в которой m строк и n столбцов, $\bar{b} \in Z^m$, $\bar{d}^- \in Z^n$ и $\bar{d}^+ \in Z^n$. 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 $\bar{c} \leqslant 0$. Для каждого индекса $i \in \{1, 2, \ldots, n\}$ такого, что $\bar{c}_i > 0$

а) в векторе $\bar{c}$ i-ую компоненту умножим на −1;

б) в матрице $\bar{A}$ i-ый столбец умножим на −1;

в) в каждом из векторов $\bar{d}^+$ и $\bar{d}^-$ i-ую компоненту умножим на −1;

г) i-ую компоненту вектора $\bar{d}^-$ и i-ую компоненту вектора $\bar{d}^+$ поменяем местами.

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

$$\begin{aligned} &c^T x + a -> \max \\ &Ax = b \\ &d^- \le x , \end{aligned} \qquad (2)$$

где

$\alpha = 0$,

$x = (x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{2n+m})^T \in R^{2n+m}$,

$c = (c_1, c_2, \ldots, c_n, 0, 0, \ldots, 0)^T \in Z^{2n+m}$

и

$A = \begin{pmatrix} \bar{A} & E_{m+n} \\ E_n & \end{pmatrix}$, $b = \begin{pmatrix} \bar{b} \\ \bar{d}^+ \end{pmatrix}$, $d^- = (\overline{d_1}^-, \overline{d_2}^-, \ldots, 0, 0, \ldots, 0)^T \in Z^{2n+m}$.

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. $r = c^T x^*$. Поместим в стек S задачу (2) вместе с вектором $\Delta = d^-$.

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 $i \in \{1, 2, \ldots, n\}$

$$x_i = \begin{cases} x_i{}^*, \text{если } \bar{c}_i < 0 \\ -x_i{}^*, \text{если } \bar{c}_i \geq 0 \end{cases}$$

относительно вектора c стоимостей исходной задачи (1).

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

$$\begin{aligned} &c^T x + a -> \max \\ &Ax = b \\ &d^- \leq x, \end{aligned}$$

с вектором $\Delta \in Z^{2n+m}$. Заметим, что вектор d^- не всегда нулевой. Приведем эту задачу к канонической форме

$$\begin{aligned} &c^T x + a' -> \max \\ &Ax = b' \\ &0 \leq x, \end{aligned} \qquad (3)$$

где $a' = a + c^T d^-$, $b' = b - Ad^-$. Построим начальный базисный двойственный план (y, B), в котором $y = (0, 0, \ldots, 0) \in Rm+n$ и $B = \{j1 = n + 1, j2 = n + 2, \ldots, jn+m = 2n + m\}$. Решим задачу (3) двойственным симплекс-методом и найдем оптимальный план $\tilde{x}$.

Рассмотрим два случая в зависимости от того является план $\tilde{x}$ целочисленным или нет.

Случай 1. Пусть план $\tilde{x}$ целочисленный. По этому плану восстановим допустимый план задачи (2) следующим образом. Прибавим к плану $\tilde{x}$ вектор Δ. Полученный в результате вектор обозначим через $x\hat{}$. Если в переменной x* нет плана или $r < c^T x\hat{} + \alpha'$, то запишем в переменную x^* план $\hat{x}$, а в переменную r значение $c^T\hat{x} + \alpha'$.

Случай 2. Пусть план $\tilde{x}$ дробный. Выберем дробную компоненту $\tilde{x}_i$ из числа первых n компонент плана $\tilde{x}$. Если в переменной x^* нет плана или $\lfloor c^T\tilde{x} + \alpha' \rfloor > r$, то построим две новые задачи линейного программирования

$$
\begin{aligned}
& c^T x + a' -> \max \\
& Ax = b'' \\
& 0 \le x ,
\end{aligned}
$$

где вектор b’’ получается из вектора b’ заменой (m+i)-ой компоненты на $\lfloor \tilde{x}_i \rfloor$ и

$$
\begin{aligned}
& c^T x + a' -> \max \\
& Ax = b' \\
& d^- \le x ,
\end{aligned}
$$

где d^- — это (2n+m)-мерный вектор, который получается из нулевого вектора заменой i-ой компоненты на $\lceil \tilde{x}_i \rceil$. Обе задачи поместим в стек S вместе с вектором $\Delta + d-$.

Повторим шаг 4.

Результат работы
Тест 1

Задание:

$$x_1 + x_2 \;\rightarrow\; \max$$
$$5 \cdot x_1 + 9 \cdot x_2 \le 63$$
$$9 \cdot x_1 + 5 \cdot x_2 \le 63$$
$$1 \le x_1 \le 6$$
$$1 \le x_2 \le 6$$
$$x_1 \in \mathbb{Z},\; x_2 \;\in\; \mathbb{Z}$$

Вывод программы:

```
Оптимальный план: (4, 4)
```

Тест 2

Задание:

$$4 \cdot x1 + x2 \rightarrow \max$$
$$x_1 + 2 \cdot x_2 \;\le 4,$$
$$3 \cdot x_1 + 2 \cdot x_2 \;\le 12$$
$$0 \le x_1 \le 3$$
$$0 \le x_2 \le 3$$
$$x_1 \in \mathbb{Z},\; x_2 \;\in\; \mathbb{Z}$$

Вывод программы:

```
Оптимальный план: (3, 0)
```

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):
```

```python
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:
```

```python
    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
```

```python
    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)
```