Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Московский государственный технический университет имени Н.Э. Баумана

Лабораторная работа № 3
«Целочисленное линенйное программирование. Метод ветвей и границ»
по курсу
«Теория игр»

Студент группы ИУ9-31М Белогуров А.А.

Преподаватель Басараб М.А.

Москва, 2019

Содержание

1 Цель работы

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

2 Постановка задачи

Задача ЦЛП имеет вид:

$$F = cx \to max, \quad Ax \leq b, \quad x \geq 0.$$

Где x - искомый вектор решения, c - вектор коэффициентов целевой функции ЦФ F, A - матрица системы ограничений, b - вектор правой части системы ограничений. Требуется по ПЗ ЛП сформулировать двойственную задачу ЛП и решить ее симплекс-методом.

Метод ветвей и границ: Сначала решается задача ЛП, связанная с данной задачей ЦЛП. При этом возможны следующие варианты:

- если решение задачи ЛП не имеет решения, то и решение задачи ЦЛП не имеет решения;
- если решение задачи ЛП имеет целочисленное решение X^* , то оно будет и решением задачи ЦЛП;
- если решение задачи ЛП X^* дробное, то оно объявляется узлом ожидания, а X^* – переменной ветвления.

Узел ожидания определяет два направления ветвления:

$$X \leq \left\lfloor X^* \right\rfloor$$

$$X \geq \left\lfloor X^* \right\rfloor + 1,$$

соответствующие двум новым задачам ЦЛП, решение которых осуществляется аналогично. Таким образом, строится двоичное дерево решений. В конце процесса сравниваются между собой все найденные целочисленные решения, соответствующие терминальным вершинам дерева, и выявляется оптимальное решение исходной задачи ЦЛП.

Исходная симлекс-таблица, вариант 2:

$$c = \begin{pmatrix} 2 & 5 & 3 \end{pmatrix} \quad b = \begin{pmatrix} 6 \\ 6 \\ 2 \end{pmatrix} \quad A = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 0 \\ 0 & 0.5 & 1 \end{pmatrix}$$

3 Практическая реализация

SimplexMatrix.py

```python
from copy import deepcopy

import numpy as np
import pandas as pd

from Lab1.Conditions import FCondition, AConditionB


class SimplexMatrix:
    def __init__(self, A, b, c, f_condition=FCondition.MAX,
    ↪  a_condition_b=AConditionB.LESS_OR_EQUAL):
        """
        1) F = cx -> max, Ax <= b
        2) F = cx -> max, Ax >= b
        3) F = cx -> min, Ax <= b
        4) F = cx -> min, AX >= b
        x_i >= 0
        """
        self.A = np.array(A, dtype='float64')
        self.b = np.array(b, dtype='float64')
        self.c = np.array(c, dtype='float64')
        self.f_condition = f_condition
        self.a_condition_b = a_condition_b

    def prepare_variables(self):
        # По умолчанию : F = cx -> max, Ax <= b
        # Канонический вид x = b - A

        if self.f_condition is FCondition.MIN:
            self.c = -self.c

        if self.a_condition_b is AConditionB.GREATER_OR_EQUAL:
            self.A = -self.A
            self.b = -self.b

        self.canonic = np.hstack((self.b, self.A))
        self.f = np.insert(-self.c, 0, 0)
```

```
37          self.canonic_with_f = np.vstack((self.canonic, self.f))
38
39          self.rows, self.cols = self.canonic_with_f.shape
40
41          self.col = ['x' + str(self.cols - 1 + i) for i in range(1,
            ↪  self.rows)] + ['f']  # Свободные
42          self.row = ['s'] + ['x' + str(i) for i in range(1,
            ↪  self.cols)]  # Базис
43
44          self.resolving_row_idx = None  # Индекс разрещающей строки
45          self.resolving_col_idx = None  # Индекс разрещающего
            ↪  столбца
46          self.resolving_element = None
47
48          self.iteration = 0
49
50      def set_resolving_elements(self, resolving_row_idx,
        ↪  resolving_col_idx):
51          self.resolving_row_idx = resolving_row_idx
52          self.resolving_col_idx = resolving_col_idx
53          self.resolving_element =
            ↪  self.canonic_with_f[resolving_row_idx,
            ↪  resolving_col_idx]
54
55      def clear_resolving_elements(self):
56          self.resolving_row_idx = None
57          self.resolving_col_idx = None
58          self.resolving_element = None
59
60      def create_new_matrix(self):
61          new_matrix = deepcopy(self)
62          new_matrix.clear_resolving_elements()
63
64          for i in range(self.rows):
65              for j in range(self.cols):
66                  if (i, j) == (self.resolving_row_idx,
                    ↪  self.resolving_col_idx):
67                      new_matrix.canonic_with_f[
68                          self.resolving_row_idx,
                            ↪  self.resolving_col_idx] = 1 /
                            ↪  self.resolving_element
69                  elif i == self.resolving_row_idx:
```

```
                    new_matrix.canonic_with_f[i, j] =
                    ↪ self.canonic_with_f[i, j] /
                    ↪ self.resolving_element
                elif j == self.resolving_col_idx:
                    new_matrix.canonic_with_f[i, j] = -
                    ↪ self.canonic_with_f[i, j] /
                    ↪ self.resolving_element
                else:
                    new_matrix.canonic_with_f[i, j] =
                    ↪ self.canonic_with_f[i, j] -
                    ↪ self.canonic_with_f[i,
                    ↪ self.resolving_col_idx] *
                    ↪ self.canonic_with_f[self.resolving_row_idx,
                    ↪ j] \ self.resolving_element

        new_matrix.canonic =
        ↪ new_matrix.canonic_with_f[0:new_matrix.rows - 1, :]
        new_matrix.f = new_matrix.canonic_with_f[new_matrix.rows -
        ↪ 1, :]

        new_matrix.col[self.resolving_row_idx],
        ↪ new_matrix.row[self.resolving_col_idx] =
        ↪ self.row[self.resolving_col_idx],
        ↪ \self.col[self.resolving_row_idx]
        new_matrix.iteration += 1

        print(new_matrix)

        return new_matrix

    def __repr__(self):
        df = pd.DataFrame(self.canonic_with_f, columns=self.row,
        ↪ index=self.col)
        return str(df) + "\n"
```

Conditions.py

```
from enum import Enum


```

```python
class FCondition(Enum):
    MIN = 1
    MAX = 2


def opposite_f_condition(cond: FCondition):
    if cond is FCondition.MIN:
        return FCondition.MAX
    elif cond is FCondition.MAX:
        return FCondition.MIN
    else:
        raise ValueError("Unknown f condition")


class AConditionB(Enum):
    LESS_OR_EQUAL = 1, "<="
    GREATER_OR_EQUAL = 2, ">="


def opposite_a_condition_b(cond: AConditionB):
    if cond is AConditionB.LESS_OR_EQUAL:
        return AConditionB.GREATER_OR_EQUAL
    elif cond is AConditionB.GREATER_OR_EQUAL:
        return AConditionB.LESS_OR_EQUAL
    else:
        raise ValueError("Unknown a condition b")
```

BranchAndBoundSimplexMethod.py

```python
import math

import numpy as np

from Lab1.Conditions import FCondition, AConditionB
from Lab1.SimplexMatrix import SimplexMatrix
from Lab1.SimplexMethod import SimplexMethod


def is_none(array):
    return list(map(lambda x: x is None, array))
```

```
12
13
14  def is_nan(array):
15      return np.isnan(array)
16
17
18  def idx_of_none_int(numbers):
19      not_is_int = lambda x: not x.is_integer()
20      idx_list = np.where(list(map(not_is_int, numbers)))[0]
21      if len(idx_list) == 0:
22          return -1
23      else:
24          return idx_list[0]
25
26
27  class BranchAndBoundSimplexMethod(SimplexMethod):
28      def __init__(self, matrix: SimplexMatrix):
29          self.current_matrix = matrix
30          super().__init__(matrix)
31
32      def start(self):
33          simplex_method = SimplexMethod(self.current_matrix)
34          simplex_method.start()  # Находим первое решение
35          while True:
36              last_iteration = simplex_method.iterations[-1]
37              idx_of_non_int =
                ↪  idx_of_none_int(last_iteration.canonic[:, 0])
38              if idx_of_non_int == -1:
39                  # Все переменные целочисленные -> найдено решение
40                  self.result = simplex_method.result
41                  return self.result
42
43              # Разделяем на две ветви и ищем оптимальное решение
44              non_int_number =
                ↪  last_iteration.canonic[idx_of_non_int, 0]
45              left_branch_number = math.floor(non_int_number)
46              right_branch_number = math.floor(non_int_number) + 1
47
48              # Номер переменной, по которой будет проходить
                ↪  ветвление
49              # ('x1' -> 0)
```

```
            element_of_branching =
            ↪  int(last_iteration.col[idx_of_non_int][1:]) - 1
            new_vector = np.array([0] * len(last_iteration.c))
            new_vector[element_of_branching] = 1

            if last_iteration.f_condition != FCondition.MAX or
            ↪  last_iteration.a_condition_b !=
            ↪  AConditionB.LESS_OR_EQUAL:
                raise ValueError("Unknown conditions!")

            left_simplex_method = None
            left_simplex_result = None

            right_simplex_method = None
            right_simplex_result = None

            try:
                print("Старт симплекс метода для левой ветви:\n")
                A_left = np.vstack((last_iteration.A, new_vector))
                b_left = np.vstack((last_iteration.b,
                ↪  left_branch_number))
                left_matrix = SimplexMatrix(A_left, b_left,
                ↪  last_iteration.c)
                left_simplex_method = SimplexMethod(left_matrix)
                left_simplex_result = left_simplex_method.start()
            except RuntimeError:
                print("Левая ветвь не имеет решения!")

            try:
                print("Старт симплекс метода для правой ветви:\n")
                A_right = np.vstack((last_iteration.A,
                ↪  -new_vector))
                b_right = np.vstack((last_iteration.b,
                ↪  -right_branch_number))
                right_matrix = SimplexMatrix(A_right, b_right,
                ↪  last_iteration.c)
                right_simplex_method = SimplexMethod(right_matrix)
                right_simplex_result =
                ↪  right_simplex_method.start()
            except RuntimeError:
                print("Правая ветвь не имеет решения!")

```

```
83                print("Левая ветвь [{}] : Правая ветвь
                  ↪  [{}]".format(left_simplex_result,
                  ↪  right_simplex_result))
84                result = np.array([left_simplex_result,
                  ↪  right_simplex_result], dtype=np.float64)
85                bool_result = np.logical_or(is_nan(result),
                  ↪  is_none(result))
86
87                if any(bool_result):
88                    if all(bool_result):
89                        # Оба решения не существуют, берем родителя
90                        self.result = simplex_method.result
91                        return result
92                    else:
93                        # Левая или правая ветвь не имеют решения
94                        error_element_idx =
                          ↪  np.where(bool_result)[0][0]
95                        # Выбираем существующее
96                        self.result = result[(error_element_idx + 1) %
                          ↪  2]
97
98                    return self.result
99                elif left_simplex_result > right_simplex_result:
100                   # Выбираем левую ветвь
101                   simplex_method = left_simplex_method
102               else:
103                   # Выбираем правую ветвь
104                   simplex_method = right_simplex_method
```

BruteForceMethod.py

```
1   import numpy as np
2   import math
3   import itertools
4
5   from Lab1.SimplexMatrix import SimplexMatrix
6
7
8   class BruteForceMethod():
9       def __init__(self, matrix: SimplexMatrix):
```

```
10          self.A = matrix.A
11          self.b = matrix.b
12          self.c = matrix.c
13
14          self.result = None
15
16      def start(self):
17          print("Start brute force method:")
18          A_transpose = np.transpose(self.A)
19          b_transpose = np.transpose(self.b)[0]
20          values = []
21          for (row_idx, row) in enumerate(A_transpose):
22              elem_values = b_transpose / row
23              elem_values[np.isinf(elem_values)] = 0
24              max_value = 0 if np.max(elem_values) < 0 else
                ↪  np.max(elem_values)
25              if max_value == 0:
26                  values.append([0])
27              else:
28                  # Сами добавляем + 1, так как range не включает
                    ↪  верхнюю границу
29                  values.append(list(range(math.ceil(max_value) +
                    ↪  1)))
30
31          value_combinations = list(itertools.product(*values))
32          brute_force_result = {}
33          for combination in value_combinations:
34              np_combination = np.array(combination)
35              condition = True
36              for (row_idx, row) in enumerate(self.A):
37                  condition &= (np.sum(row * np_combination) <=
                    ↪  self.b[row_idx])[0]
38
39              if condition:
40                  f_value = np.sum(np.array(self.c) *
                    ↪  np_combination)
41                  brute_force_result[f_value] = np_combination
42
43          for f_value, combination in brute_force_result.items():
44              print("F = {}, X = {}".format(f_value, combination))
45
46          max_f = max(brute_force_result, key=int)
```

```
47          print("\nmax(F) = {}, X = {}".format(max_f,
            ↪  brute_force_result[max_f]))
48
49          self.result = max_f
50          return self.result
```

4 Результаты

Решение задачи ЦЛП методом ветвей и границ:

```
Start simplex method
      s   x1   x2   x3
x4  6.0  2.0  1.0  2.0
x5  6.0  1.0  2.0  0.0
x6  2.0  0.0  0.5  1.0
f   0.0 -2.0 -5.0 -3.0

Замена базисной переменной x1 на свободную x4
      s   x4   x2   x3
x1  3.0  0.5  0.5  1.0
x5  3.0 -0.5  1.5 -1.0
x6  2.0 -0.0  0.5  1.0
f   6.0  1.0 -4.0 -1.0

Замена базисной переменной x3 на свободную x6
      s   x4   x2   x6
x1  1.0  0.5  0.0 -1.0
x5  5.0 -0.5  2.0  1.0
x3  2.0 -0.0  0.5  1.0
f   8.0  1.0 -3.5  1.0

Замена базисной переменной x2 на свободную x5
        s     x4    x5    x6
x1   1.00  0.500 -0.00 -1.00
x2   2.50 -0.250  0.50  0.50
x3   0.75  0.125 -0.25  0.75
f   16.75  0.125  1.75  2.75

Старт симплекс метода для левой ветви:

Start simplex method
      s   x1   x2   x3
x4  6.0  2.0  1.0  2.0
x5  6.0  1.0  2.0  0.0
x6  2.0  0.0  0.5  1.0
x7  2.0  0.0  1.0  0.0
f   0.0 -2.0 -5.0 -3.0
```

```
38
39  Замена базисной переменной x1 на свободную x4
40        s   x4   x2   x3
41  x1  3.0  0.5  0.5  1.0
42  x5  3.0 -0.5  1.5 -1.0
43  x6  2.0 -0.0  0.5  1.0
44  x7  2.0 -0.0  1.0  0.0
45  f   6.0  1.0 -4.0 -1.0
46
47  Замена базисной переменной x3 на свободную x6
48        s   x4   x2   x6
49  x1  1.0  0.5  0.0 -1.0
50  x5  5.0 -0.5  2.0  1.0
51  x3  2.0 -0.0  0.5  1.0
52  x7  2.0  0.0  1.0 -0.0
53  f   8.0  1.0 -3.5  1.0
54
55  Замена базисной переменной x2 на свободную x7
56         s   x4   x7   x6
57  x1   1.0  0.5 -0.0 -1.0
58  x5   1.0 -0.5 -2.0  1.0
59  x3   1.0 -0.0 -0.5  1.0
60  x2   2.0  0.0  1.0 -0.0
61  f   15.0  1.0  3.5  1.0
62
63  Старт симплекс метода для правой ветви:
64
65  Start simplex method
66        s   x1   x2   x3
67  x4  6.0  2.0  1.0  2.0
68  x5  6.0  1.0  2.0  0.0
69  x6  2.0  0.0  0.5  1.0
70  x7 -3.0  0.0 -1.0  0.0
71  f   0.0 -2.0 -5.0 -3.0
72
73  Замена базисной переменной x2 на свободную x7
74         s   x1   x7   x3
75  x4   3.0  2.0  1.0  2.0
76  x5   0.0  1.0  2.0  0.0
77  x6   0.5  0.0  0.5  1.0
78  x2   3.0 -0.0 -1.0 -0.0
79  f   15.0 -2.0 -5.0 -3.0
```

```

Замена базисной переменной x1 на свободную x5
       s   x5   x7   x3
x4   3.0 -2.0 -3.0  2.0
x1   0.0  1.0  2.0  0.0
x6   0.5 -0.0  0.5  1.0
x2   3.0  0.0 -1.0  0.0
f   15.0  2.0 -1.0 -3.0

Замена базисной переменной x7 на свободную x1
       s    x5    x1   x3
x4   3.0 -0.50  1.50  2.0
x7   0.0  0.50  0.50  0.0
x6   0.5 -0.25 -0.25  1.0
x2   3.0  0.50  0.50  0.0
f   15.0  2.50  0.50 -3.0

Замена базисной переменной x3 на свободную x7
     s   x5   x1   x7
x4 NaN -inf -inf -inf
x3 NaN  inf  inf  inf
x6 NaN -inf -inf -inf
x2 NaN  NaN  NaN  NaN
f  NaN  inf  inf  inf

Левая ветвь [15.0] : Правая ветвь [nan]
```

Решений задачи ЦЛП перебором:

```
F = 0.0, X = [0 0 0]
F = 3.0, X = [0 0 1]
F = 6.0, X = [3 0 0]
F = 5.0, X = [1 0 1]
F = 8.0, X = [1 0 2]
F = 10.0, X = [1 1 1]
F = 13.0, X = [0 2 1]
F = 15.0, X = [1 2 1]
F = 2.0, X = [1 0 0]
F = 7.0, X = [2 0 1]
F = 12.0, X = [1 2 0]
```

```
F = 4.0, X = [2 0 0]
F = 9.0, X = [2 1 0]
F = 14.0, X = [2 2 0]

max(F) = 15.0, X = [1 2 1]
```

Результаты обоих методов совпадают:

$$F = 15, \quad x = (1, 2, 1)$$