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

Лабораторная работа №6. Вариант 1. «Методы решения задач линейного и линейного целочисленного программирования» по курсу «Методы оптимизации»

Студент группы ИУ9-82 Иванов Г.М

Преподаватель Каганов Ю.Т.

Москва, 2019

Содержание

1 Цель работы

1. Изучение симплекс-метода линейного программирования и методов линейного целочисленного программирования.

2. Разработка программы реализации алгоритма симплекс-метода линейного программирования и алгоритмов линейного целочисленного программирования.

3. Решение задачи линейного и линейного целочисленного программирования.

2 Постановка задачи

Дано: 1 Вариант. Функция:

$$f(x) = x_1 - x_2 + x_3. \tag{1}$$

Функции ограничений:

$$\begin{cases} -x_1 + 2x_2 - x_3 = 4 \\ 3x_1 + x_2 + x_4 = 14 \\ x_1, ..., x_4 \geq 0 \end{cases} \tag{2}$$

2.1 Задача 6.1

1. Найти условный максимум для задачи линейного программирования симплекс-методом Дж. Данцига.

2. Реализовать алгоритмы с помощью языка программирования высокого уровня.

2.2 Задача 6.2

1. Найти условный максимум для задачи линейного целочисленного программирования методом «ветвей и границ».

2. Реализовать алгоритмы с помощью языка программирования высокого уровня.

3 Исследование

Найдем условный максимум для функции:

$$f(x) = x_1 - x_2 + x_3 \tag{3}$$

с помощью сервиса WolframAlpha.com:

$$max(f(x)) = 10, \quad (x_1, x_2, x_3, x_4) = (0, 14, 24, 0) \tag{4}$$

3.1 Задача 6.1

3.1.1 Симлекс-метод

Это алгоритм решения оптимизационной задачи линейного программирования путём перебора вершин выпуклого многогранника в многомерном пространстве.

Сущность метода: построение базисных решений, на которых монотонно убывает линейный функционал, до ситуации, когда выполняются необходимые условия локальной оптимальности.

3.2 Задача 6.2

3.2.1 Метод ветвей и границ

Общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Общая идея метода может быть описана на примере поиска минимума функции $f(x)$ на множестве допустимых значений переменной

x. Функция f и переменная x могут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Так в данном случае необходимо решить задачу целочисленного линейного программирования, то необходимо изменить исходную функцию, так как её решением при первой итерации являются целые числа.

Новая функция для **Задачи 6.2** будет выглядеть следующим образом:

$$f(x) = -x_1 + 11x_2 - 1x_3 + 5x_4 \tag{5}$$

Функции ограничений не изменяются.

4 Практическая реализация

Все методы были реализованы на языке программирования **Python**.

Листинг 1. Симлекс-метод.

```
class MatrixSimplex:
    def __init__(self, A, b, c, B_idx):
                self.A = np.matrix(A)
                self.b = np.matrix(b)
                self.c = np.matrix(c)

                self.B_idx = B_idx
                self.N_idx = [i for i in range(self.A.shape[1]) if i not in
                ↪  self.B_idx]

        def get_B(self):
                return self.A[:, self.B_idx]

        def get_Cb(self):
                return self.c[:, self.B_idx]

        def get_Cn(self):
                return self.c[:, self.N_idx]

        def get_N(self):
                return self.A[:, self.N_idx]

        def computeZ(self, InvB, N):
                # z = Cb*InvB*b - (Cb*InvB*N - Cn)*Xn
                Cb = self.get_Cb()
                Cb_InvB = np.dot(Cb, InvB)

                Cb_InvB_b = np.dot(Cb_InvB, self.b.T)

                Cb_InvB_N = np.dot(Cb_InvB, N)
                bracket_term = np.subtract(Cb_InvB_N, self.get_Cn())
                return Cb_InvB_b, -1 * bracket_term

        def computeXb(self, InvB, N):
                return np.dot(InvB, self.b.T)

        def is_optimal(self, z):
                for x in np.nditer(z):
                        if x > 0:
                                return False
```

```
40                  return True
41
42          def compute_entering_variable(self, z):
43                  max_ascent = 0
44                  index = -1
45                  it = np.nditer(z.T, flags=['f_index'])
46                  while not it.finished:
47                          if it[0] > max_ascent:
48                                  max_ascent = it[0]
49                                  index = it.index
50                          it.iternext()
51                  if index is not -1:  # else returns None
52                          return self.N_idx[index]
53
54          def replace(self, ev_idx, dv_idx):
55                  for i in range(len(self.B_idx)):
56                          if self.B_idx[i] == dv_idx:
57                                  self.B_idx[i] = ev_idx
58                                  break
59
60                  self.N_idx = [i for i in range(self.A.shape[1]) if i not in
                    ↪  self.B_idx]
61
62          def calculate_value(self, z, Xb):
63                  print("z = ", z[0, 0])
64                  for i in range(self.c.shape[1]):
65                          index = 0
66                          in_basis = False
67                          for j in self.B_idx:
68                                  if i == j:
69                                          print("x[{}] = {}".format(j,
                                            ↪  Xb[index, 0]))
70                                          in_basis = True
71                                  index += 1
72                          if not in_basis:
73                                  print("x[{}] = {}".format(i, 0))
74                  print("")
75                  return
76
77          def do_simplex(self):
78                  print("\n######## Start Simplex ###########\n")
79                  while True:
80                          InvB = np.linalg.inv(self.get_B())
81                          N = self.get_N()
82
83                          z, z_var = self.computeZ(InvB, N)
84
85                          Xb = self.computeXb(InvB, N)
86
```

```
                        if self.is_optimal(z_var):
                                print("Optimal solution found..")
                                self.calculate_value(z, Xb)
                                return z

                        ev_idx = self.compute_entering_variable(z_var)
                        if ev_idx is None:
                                print("Invalid")
                                return 0

                        ev = self.A[:, ev_idx]
                        dv_idx = self.min_ratio_test(Xb, ev)

                        if dv_idx is None:
                                print("Invalid")
                                return 0

                        self.replace(ev_idx, dv_idx)
```

Листинг 2. Метод ветвей и границ.

```
class Node:
        def __init__(self, x_bounds=[], freeze_var_list=[], index=0,
        ↪  upper_or_lower=0):
                self._x_bounds = x_bounds
                self._freeze_var_list = freeze_var_list
                self._index = index
                self._upper_or_lower = upper_or_lower

        def freeze_lower_var(self, index, val):
                self._x_bounds[index] = (None, val)
                self._freeze_var_list.append(index)

        def freeze_upper_var(self, index, val):
                self._x_bounds[index] = (val, None)
                self._freeze_var_list.append(index)

        def freeze_var(self, index, val):
                self._x_bounds[index] = (val, val)
                self._freeze_var_list.append(index)

        def set_lp_res(self, res):
                self._res = res

        def check_integer_var_all_solved(self, m):
                return True if m == len(self._freeze_var_list) else False
```

```
25
26
27  def check_all_integers(list):
28          is_integer = True
29          for i in range(len(list) - 1):
30                  if not is_int(list[i]):
31                          is_integer = False
32                          break
33
34          return is_integer
35
36
37  def calculate(A, b, c):
38          global node_counter
39
40          x_bounds = [(0, None) for i in range(len(c))]
41
42          print("\n######## Start B & B ###########\n")
43
44          node = Node(copy.deepcopy(x_bounds), [], node_counter)
45
46          node_counter += 1
47          res = solve_LP(x_bounds, A, b, c)
48
49          if check_all_integers(res.x):
50                  print_result(res)
51                  return res
52
53
54          lower = floor(res['x'][integer_var[0]])
55          upper = lower + 1
56
57          lower_node = Node(copy.deepcopy(x_bounds), [], node_counter, 1)
58          lower_node.freeze_lower_var(integer_var[0], lower)
59          add_dangling_node(lower_node, A, b, c)
60
61          node_counter += 1
62
63          upper_node = Node(copy.deepcopy(x_bounds), [], node_counter, 2)
64          upper_node.freeze_upper_var(integer_var[0], upper)
65          add_dangling_node(upper_node, A, b, c)
66
67          node_counter += 1
68
69          while len(dangling_nodes) > 0:
70
71                  index = np.argmin(dangling_nodes_obj)
72
73                  x_b = dangling_nodes[index]._x_bounds
```

```
                frez = dangling_nodes[index]._freeze_var_list
                res = dangling_nodes[index]._res
                frez_var_index = len(frez)

                u_or_l = dangling_nodes[index]._upper_or_lower
                arbitrary_node = Node(copy.deepcopy(x_b),
                ↪  copy.deepcopy(frez), node_counter, copy.deepcopy(u_or_l))
                u_or_l_b = lower - 1 if (u_or_l == 1) else upper + 1
                arbitrary_node.freeze_var(integer_var[frez_var_index - 1],
                ↪  u_or_l_b)
                x_b_arbi = arbitrary_node._x_bounds
                if check_bounds(x_b_arbi, integer_var[frez_var_index - 1],
                ↪  u_or_l, x_bounds):
                        add_dangling_node(arbitrary_node, A, b, c)
                else:
                        print("arbitrary Node infeasibile: ",
                        ↪  arbitrary_node._index)

                node_counter += 1

                lower = floor(res['x'][integer_var[frez_var_index]])
                upper = lower + 1

                lower_node = Node(copy.deepcopy(x_b), copy.deepcopy(frez),
                ↪  node_counter, 1)
                lower_node.freeze_lower_var(integer_var[frez_var_index],
                ↪  lower)
                add_dangling_node(lower_node, A, b, c)

                node_counter += 1

                upper_node = Node(copy.deepcopy(x_b), copy.deepcopy(frez),
                ↪  node_counter, 2)
                upper_node.freeze_upper_var(integer_var[frez_var_index],
                ↪  upper)
                add_dangling_node(upper_node, A, b, c)

                node_counter += 1

                # убрать break в записке
                # if check_all_integers():
                break

        result = get_max_integer_node()
        print_result(result._res)


def is_int(n):
        return int(n) == float(n)
```

```
115
116
117 def solve_LP(x_bounds, A_eq, b_eq, c):
118         return Simplex(c, A_eq=A_eq, b_eq=b_eq, bounds=x_bounds)
119
120
121 def print_result(result):
122         x_list = result.x
123         print("z = {}".format(result.fun))
124
125         for i, item in enumerate(x_list):
126                 print("x[{}] = {}".format(i, item))
127
128 def get_max_integer_node():
129         global dangling_nodes
130         global dangling_nodes_obj
131         global result_Node
132
133         integer_nodes = []
134         for node in dangling_nodes:
135                 if check_all_integers(node._res.x):
136                         integer_nodes.append(node)
137
138         max_node_result = float("-INF")
139         max_node = None
140         for node in integer_nodes:
141                 if node._res.fun >= max_node_result:
142                         max_node = node
143                         max_node_result = node._res.fun
144
145         return max_node
146
147 def add_dangling_node(node, A, b, c):
148         global z_star
149         global dangling_nodes
150         global dangling_nodes_obj
151         global result_Node
152
153         res = solve_LP(node._x_bounds, A, b, c)
154         if check_feasibility(res) and res['fun'] > z_star:
155                 node.set_lp_res(res)
156
157                 if check_all_integers(res.x):
158                         dangling_nodes_obj.append(res['fun'])
159                         dangling_nodes.append(node)
160                 elif len(dangling_nodes) == 0:
161                         dangling_nodes_obj.append(res['fun'])
162                         dangling_nodes.append(node)
163                 else:
```

```
164                         return
165
166                 if node.check_integer_var_all_solved(len(integer_var)):
167                         z_star = res['fun']
168                         result_Node = node
169                 return True
170         else:
171                 return False
```

5 Результаты.

Программы, представленные в **Листинге 1** и **Листинге 2** дали следующие результаты:

Листинг 5. Результаты выполнения программ.

```
Start Simplex method:
    z =  10.0
    x[0] = 0
    x[1] = 14.0
    x[2] = 24.0
    x[3] = 0

Start B&B method
    [(None, 3), (None, 3), (0, None), (0, None)]
    z = 56.0
    x[0] = 2.0
    x[1] = 3.0
    x[2] = 0.0
    x[3] = 5.0
```

Проверка полученных значений для ограничений:

$$\begin{cases} -x_1 + 2x_2 - x_3 = 4 \\ 3x_1 + x_2 + x_4 = 14 \\ x_1, ..., x_4 \geq 0 \end{cases} \tag{6}$$

Симплекс-метод:

$$\begin{cases} -1 \times 0 + 2 \times 14 - 1 \times 24 + 0 \times 0 = 4 \\ 3 \times 0 + 1 \times 14 + 0 \times 0 + 1 \times 0 = 14 \\ x_1, ..., x_4 \geq 0 \end{cases} \tag{7}$$

Проверка прошла успешно.

Метод ветвей и границ:

$$\begin{cases} -1 \times 2 + 2 \times 3 - 1 \times 0 + 0 \times 5 = 4 \\ 3 \times 2 + 1 \times 3 + 0 \times 0 + 1 \times 5 = 14 \\ -\infty \leq x_1 \leq 3, \quad -\infty \leq x_2 \leq 3, \quad 0 \leq x_3 \leq \infty, \quad 0 \leq x_4 \leq \infty \end{cases} \tag{8}$$

Проверка прошла успешно.