Целочисленное линейное программирование. Метод ветвей и границ

Цель: изучить постановку задачи целочисленного линейного программирования (ЦЛП); получить навыки решения задачи ЦЛП методом ветвей и границ (МВГ).

Вариант №8

Найти целочисленное решение $\mathbf{x} = (x_1, x_2, x_3)^{\mathrm{т}}$ следующей задачи:

$$F = \mathbf{cx} \to \max;$$

$$\mathbf{Ax} \le \mathbf{b};$$

$$x_1, x_2, x_3 \ge 0.$$

$\mathbf{c} = (1\ 5\ 5)$

$$\mathbf{A} = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0,5 & 4 \end{pmatrix}$$

$\mathbf{b}^{\mathrm{т}} = (6\ 5\ 5)$

Результаты работы:

```
1) Результат полного перебора:
Лучшее целочисленное решение (перебор): x = [0. 5. 0.]  F = 25.0
```

Рисунок 1 Результат перебора

```
2) Решение релаксации LP (симплекс):
x_LP = [0.    5.    0.625]  F_LP = 28.125

Начальная (исходная) симплекс-таблица:
[[ 4.   1.   1.   1.   0.   0.   6. ]
 [ 1.   1.   0.   0.   1.   0.   5. ]
 [ 0.   0.5  4.   0.   0.   1.   5. ]
 [-1.  -5.  -5.   0.   0.   0.   0. ]]

Шаги симплекс-метода (таблицы):

Таблица 0:
 [[ 4.   1.   1.   1.   0.   0.   6. ]
 [ 1.   1.   0.   0.   1.   0.   5. ]
 [ 0.   0.5  4.   0.   0.   1.   5. ]
 [-1.  -5.  -5.   0.   0.   0.   0. ]]

Таблица 1:
 [[ 1.    0.25  0.25  0.25  0.    0.    1.5 ]
 [ 0.    0.75 -0.25 -0.25  1.    0.    3.5 ]
 [ 0.    0.5   4.    0.    0.    1.    5.  ]
 [ 0.   -4.75 -4.75  0.25  0.    0.    1.5 ]]

Таблица 2:
 [[ 1.        0.        0.333333  0.333333 -0.333333  0.        0.333333]
 [ 0.        1.       -0.333333 -0.333333  1.333333  0.        4.666667]
 [ 0.        0.        4.166667  0.166667 -0.666667  1.        2.666667]
 [ 0.        0.       -6.333333 -1.333333  6.333333  0.       23.666667]]

Таблица 3:
 [[ 1.    0.    0.    0.32 -0.28 -0.08  0.12]
 [ 0.    1.    0.   -0.32  1.28  0.08  4.88]
 [ 0.    0.    1.    0.04 -0.16  0.24  0.64]
 [ 0.    0.    0.   -1.08  5.32  1.52 27.72]]

Таблица 4:
 [[ 3.125  0.     0.     1.    -0.875 -0.25   0.375]
 [ 1.     1.     0.     0.     1.     0.     5.   ]
 [-0.125  0.     1.     0.    -0.125  0.25   0.625]
 [ 3.375  0.     0.     0.     4.375  1.25  28.125]]
```

Рисунок 2 Симплекс

```
3) Метод ветвей и границ (лог операций):

--- Обработка узла 1 (root), доп. ограничения: [] ---
Начальная симплекс-таблица для узла:
[[ 4.   1.   1.   1.   0.   0.   6. ]
 [ 1.   1.   0.   0.   1.   0.   5. ]
 [ 0.   0.5  4.   0.   0.   1.   5. ]
 [-1.  -5.  -5.   0.   0.   0.   0. ]]
Решение релаксации: [0.    5.    0.625]  F = 28.125
Переменная для ветвления: x3 = 0.625000, floor=0, ceil=1

--- Обработка узла 2 (root->x3<=0), доп. ограничения: [(array([0., 0., 1.]), np.float64(0.0), '<=')] ---
Начальная симплекс-таблица для узла:
[[ 4.   1.   1.   1.   0.   0.   0.   6. ]
 [ 1.   1.   0.   0.   1.   0.   0.   5. ]
 [ 0.   0.5  4.   0.   0.   1.   0.   5. ]
 [ 0.   0.   1.   0.   0.   0.   1.   0. ]
 [-1.  -5.  -5.   0.   0.   0.   0.   0. ]]
Решение релаксации: [0. 5. 0.]  F = 25.0
Найдена целая точка релаксации: [0 5 0]
Обновлён лучший целочисленный: {'val': np.float64(25.0), 'x': array([0, 5, 0])}

--- Обработка узла 3 (root->x3>=1), доп. ограничения: [(array([0., 0., 1.]), np.float64(1.0), '>=')] ---
LP недоступен: Infeasible due to negative RHS (unexpected)
```

Рисунок 3 Метод ветвей и границ

```
4) Найденное целочисленное решение (BnB):
Лучший целочисленный (BnB) x = [0 5 0]  F = 25.0
```

Рисунок 4 Целочисленное решение

```
5) Сравнение с округлениями решения LP:
LP solution: [0.    5.    0.625]
Round down: [0. 5. 0.]  F = 25.0  Feasible: True
Round up  : [0. 5. 1.]  F = 30.0  Feasible: False
```

Рисунок 5 Сравнение с округлениями решения LP

```
Итоги:
 - Полный перебор дал: x = [0. 5. 0.]  F = 25.0
 - Релаксация (LP) дала: x_LP = [0.    5.    0.625]  F_LP = 28.125
 - BnB дал: x_BnB = [0 5 0]  F_BnB = 25.0
```

Рисунок 6 Итог

Листинг программы:

```python
import numpy as np

c = np.array([1.0, 5.0, 5.0])
A = np.array([[4.0, 1.0, 1.0],
              [1.0, 1.0, 0.0],
              [0.0, 0.5, 4.0]])
b = np.array([6.0, 5.0, 5.0])

n = c.size
m = b.size

def compute_variable_upper_bounds(A, b):
    bounds = []
    for j in range(A.shape[1]):
        vals = []
        for i in range(A.shape[0]):
            if A[i,j] > 1e-12:
                vals.append(b[i] / A[i,j])
        if vals:
            bounds.append(int(np.floor(min(vals))))
        else:
            bounds.append(0)
    return bounds

class SimplexSolver:
    def __init__(self, A, b, c):
        self.A = A.astype(float)
        self.b = b.astype(float)
        self.c = c.astype(float)
        self.m, self.n = A.shape[0], A.shape[1]
        self.table = None
        self.basic = None
        self.nonbasic = None
        self.tableaus = []
        self._build_initial_tableau()
```

```python
def _build_initial_tableau(self):
    m, n = self.m, self.n
    T = np.zeros((m+1, n+m+1))
    T[0:m, 0:n] = self.A
    T[0:m, n:n+m] = np.eye(m)
    T[0:m, -1] = self.b
    T[m, 0:n] = -self.c
    T[m, -1] = 0.0
    self.table = T
    self.basic = list(range(n, n+m))
    self.nonbasic = list(range(0, n))
    self.tableaus.append(self.table.copy())

def _is_optimal(self):
    return np.all(self.table[-1, :-1] >= -1e-9)

def _select_pivot(self):
    reduced = self.table[-1, :-1]
    for j, val in enumerate(reduced):
        if val < -1e-9:
            pivot_col = j
            break
    else:
        return None, None
    col = self.table[0:self.m, pivot_col]
    rhs = self.table[0:self.m, -1]
    ratios = []
    for i in range(self.m):
        if col[i] > 1e-12:
            ratios.append((rhs[i] / col[i], i))
    if not ratios:
        return pivot_col, None
    ratios.sort(key=lambda x: (x[0], x[1]))
    pivot_row = ratios[0][1]
    return pivot_col, pivot_row

def _pivot(self, pivot_row, pivot_col):
    T = self.table
    piv = T[pivot_row, pivot_col]
    T[pivot_row, :] = T[pivot_row, :] / piv
    for i in range(T.shape[0]):
        if i != pivot_row:
            factor = T[i, pivot_col]
            T[i, :] = T[i, :] - factor * T[pivot_row, :]
    self.basic[pivot_row] = pivot_col
    self.tableaus.append(self.table.copy())

def solve(self, max_iters=200):
    it = 0
    while (not self._is_optimal()) and it < max_iters:
        pivot_col, pivot_row = self._select_pivot()
        if pivot_col is None:
            break
        if pivot_row is None:
            raise Exception("Unbounded LP.")
        self._pivot(pivot_row, pivot_col)
        it += 1
    sol = np.zeros(self.n + self.m)
    for i, colidx in enumerate(self.basic):
        if colidx < self.n + self.m:
            sol[colidx] = self.table[i, -1]
    x = sol[0:self.n]
    z = self.table[-1, -1]
```

```
        return x, z, self.tableaus

bounds = compute_variable_upper_bounds(A, b)
print("Оценки верхних границ для перебора:", bounds)

brute_best_val = -1e9
brute_best_x = None
for x1 in range(bounds[0] + 1):
    for x2 in range(bounds[1] + 1):
        for x3 in range(bounds[2] + 1):
            x = np.array([x1, x2, x3], dtype=float)
            if np.all(A.dot(x) <= b + 1e-9):
                val = c.dot(x)
                if val > brute_best_val:
                    brute_best_val = val
                    brute_best_x = x.copy()

print("\n1) Результат полного перебора:")
print("Лучшее целочисленное решение (перебор): x =", brute_best_x, " F =",
brute_best_val)

solver = SimplexSolver(A, b, c)
x_lp, z_lp, tableaus_lp = solver.solve()
print("\n2) Решение релаксации LP (симплекс):")
print("x_LP =", np.round(x_lp,6), " F_LP =", round(z_lp,6))

print("\nНачальная (исходная) симплекс-таблица:")
np.set_printoptions(precision=6, suppress=True)
print(tableaus_lp[0])

print("\nШаги симплекс-метода (таблицы):")
for i, T in enumerate(tableaus_lp):
    print(f"\nТаблица {i}:\n", T)

class BnBNode:
    def __init__(self, extra_constraints=None, name="root"):
        self.extra_constraints = extra_constraints or []
        self.name = name

def solve_lp_with_extra_constraints(A, b, c, extra_constraints):
    A_ext = A.copy()
    b_ext = b.copy()
    for row, rhs, sense in extra_constraints:
        if sense == '<=':
            A_ext = np.vstack([A_ext, np.array(row, dtype=float)])
            b_ext = np.append(b_ext, float(rhs))
        elif sense == '>=':
            A_ext = np.vstack([A_ext, -np.array(row, dtype=float)])
            b_ext = np.append(b_ext, -float(rhs))
        else:
            raise ValueError("sense must be '<=' or '>='")
    solver = SimplexSolver(A_ext, b_ext, c)
    try:
        x_lp, z_lp, tableaus = solver.solve()
    except Exception as e:
        return None, None, None, str(e)
    if np.any(b_ext < -1e-9):
        return None, None, None, "Infeasible due to negative RHS"
    return x_lp, z_lp, tableaus, None

best_incumbent = {'val': -1e9, 'x': None}
stack = [BnBNode(extra_constraints=[], name="root")]

print("\n3) Метод ветвей и границ (лог операций):")
```

```
node_counter = 0
while stack:
    node = stack.pop()
    node_counter += 1
    print(f"\n--- Обработка узла {node_counter} ({node.name}), доп. 
ограничения: {node.extra_constraints} ---")
    x_relax, z_relax, tableaus_relax, err = 
solve_lp_with_extra_constraints(A, b, c, node.extra_constraints)
    if err:
        print("LP недоступен:", err)
        continue
    print("Начальная симплекс-таблица для узла:")
    print(tableaus_relax[0])
    print("Решение релаксации:", np.round(x_relax,8), " F =", z_relax)
    if z_relax <= best_incumbent['val'] + 1e-9:
        print("Отсечено: оценка <= лучший целочисленный.")
        continue
    frac_indices = [j for j in range(len(x_relax)) if abs(x_relax[j] - 
round(x_relax[j])) > 1e-9]
    if not frac_indices:
        print("Найдена целая точка релаксации:", 
np.round(x_relax).astype(int))
        if z_relax > best_incumbent['val'] + 1e-9:
            best_incumbent['val'] = z_relax
            best_incumbent['x'] = np.round(x_relax).astype(int)
            print("Обновлён лучший целочисленный:", best_incumbent)
        continue
    frac_parts = [(j, abs(x_relax[j] - np.floor(x_relax[j]))) for j in 
frac_indices]
    frac_parts.sort(key=lambda t: -t[1])
    j_branch = frac_parts[0][0]
    xj = x_relax[j_branch]
    lower = np.floor(xj)
    upper = np.ceil(xj)
    print(f"Переменная для ветвления: x{j_branch+1} = {xj:.6f}, 
floor={int(lower)}, ceil={int(upper)}")
    left_constraints = node.extra_constraints + [(np.eye(1, n, 
j_branch).flatten(), lower, '<=')]
    right_constraints = node.extra_constraints + [(np.eye(1, n, 
j_branch).flatten(), upper, '>=')]
    stack.append(BnBNode(extra_constraints=right_constraints, name=node.name 
+ f"->x{j_branch+1}>={int(upper)}"))
    stack.append(BnBNode(extra_constraints=left_constraints, name=node.name + 
f"->x{j_branch+1}<={int(lower)}"))

print("\n4) Найденное целочисленное решение (BnB):")
print("Лучший целочисленный (BnB) x =", best_incumbent['x'], " F =", 
best_incumbent['val'])

print("\n5) Сравнение с округлениями решения LP:")
x_lp_rounded_down = np.floor(x_lp)
x_lp_rounded_up = np.ceil(x_lp)
def is_feasible(x):
    return np.all(A.dot(x) <= b + 1e-9) and np.all(x >= -1e-9)

print("LP solution:", x_lp)
print("Round down:", x_lp_rounded_down, " F =", c.dot(x_lp_rounded_down), " 
Feasible:", is_feasible(x_lp_rounded_down))
print("Round up  :", x_lp_rounded_up, " F =", c.dot(x_lp_rounded_up), " 
Feasible:", is_feasible(x_lp_rounded_up))

print("\nИтоги:")
print(" - Полный перебор дал: x =", brute_best_x, " F =", brute_best_val)
```

```
print(" - Релаксация (LP) дала: x_LP =", np.round(x_lp,6), " F_LP =",
round(z_lp,6))
print(" - BnB дал: x_BnB =", best_incumbent['x'], " F_BnB =",
best_incumbent['val'])
```

Вывод: в ходе выполнения лабораторной работы была поставлена и решена задача целочисленного линейного программирования по МВГ.