ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра Математической кибернетики и информационных технологий

Лабораторная работа

по дисциплине

Теория оптимизации

на тему:

Решение транспортной задачи средствами языка программирования Python

Выполнил: студент гр. 3МПП1901 Галузинский И.А,

Принял: доцент к.т.н., Городничев М.Г.

Москва 2019 г.

**Содержание**

1. Постановка задачи3
2. Решение задачи 4
3. Код программам (Python)6
   1. Библиотека scipy.optimize………………………………………………………………………….6
   2. Библиотека Cvxopt.modeling…………………………………………………….……………….8

3.3 Библиотека pulp……………………………………………………………………………………….10

1. Сравнение библиотек и вывод12

Список использованных источников13

**1.Постановка задачи:**

Найти минимальную сумму с помощью указанных библиотек:

1. Pulp
2. Cvxopt
3. Optimazw

Придуман следующий граф:

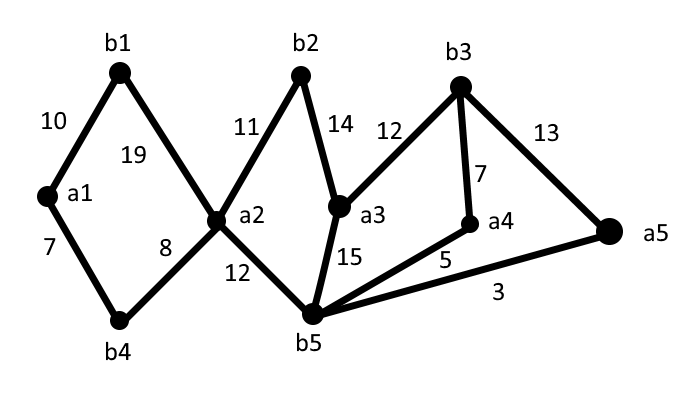


Рисунок 1. Граф «перевозка товара», -это место доставки товара.

2. Решение.

Для каждого - были взяты случайные величины, но с одним условием, чтобы выполнялось следующие условие ≥.

Далее будет произведён расчёт наиболее коротких путей от складов до заказчиков .

От a₁-bᵢ:

B₁a₁=a₁b₁=10;

B₂a₁=a₁b₄a₂b₂=7+8+11=26;

B₃a₁=a₁b₄a₂b₅a₄b₃=7+8+12+5+7=39;

B₄a₁=a₁b₄=7;

B₅a₁=a₁b₄a₂b₅=7+8+12=27;

От a₂-bᵢ:

B₁a₂=a₂b₁=19;

B₂a₂=a₂b₂=11;

B₃a₂=a₂b₅a₄b₃=12+5+7=24;

B₄a₂=a₂b₄=8;

B₅a₂=a₂b₅=12;

От a₃-bᵢ:

B₁a₃=a₃b₂a₂b₁=14+11+19=44;

B₂a₃=a₃b₂=14;

B₃a₃=a₃b₃=12;

B₄a₃=a₃b₂b₄=14+11+8=33;

B₅a₃=a₃b₅=15;

От a₄-bᵢ:

B₁a₄=a₄b₅a₂b₁=5+12+19=36;

B₂a₄=a₄b₃a₃b₂=7+12+14=33;

B₃a₄=a₄b₃=7;

B₄a₄=a₄b₅a₂b=5+12+8=25;

B₅a₄=a₄b₅=5;

От a₅-bᵢ:

B₁a₅=a₅b₅a₂b₁=3+12+19=34;

B₂a₅=a₅b₅a₃b₂=3+15+14=32;

B₃a₅=a₅b₃=13;

B₄a₅=a₅b₅a₂b₄=3+12+8=23;

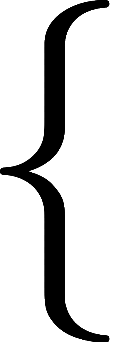
B₅a₅=a₅b₅=3;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | b₁ | b₂ | b₃ | b₄ | b₅ |
| a₁ | 10 | 26 | 39 | 7 | 27 |
| a₂ | 19 | 11 | 24 | 8 | 12 |
| a₃ | 44 | 14 | 12 | 33 | 15 |
| a₄ | 36 | 33 | 7 | 25 | 5 |
| a₅ | 34 | 32 | 13 | 23 | 3 |

Таблица 1. Иллюстрация стоимости доставки одного товара от склада aᵢ к заказчику bᵢ.

Значение b₁- b₅ и a₁ - a₅ были взяты случайными, но при одном условии, чтобы ≥.

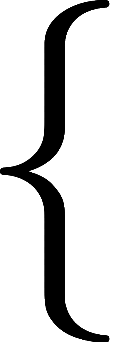
Следовательно, теперь можно составить следующие системы ограничений(равенств) и не равенств, которые помогут в будущем:

Х₀+X₁+X₂+X₃+X₄⩽ 40

X₅+X₆+X₇+X₈+X₉⩽15

X₁₀+X₁₁+X₁₂+X₁₃+X₁₄ ⩽7

X₁₅+X₁₆+X₁₇+X₁₈+X₁₉ ⩽10

X₂₀+X₂₁+X₂₂+X₂₃+X₂₄ ⩽8

X₀+X₄+X₁₀+X₁₄+X₂₀=10

X₁+X₆+X₁₁+X₁₆+X₂₁=8

X₂+X₇+X₁₂+X₁₇+X₂₂=40

X₃+X₈+X₁₃+X₁₈+X₂₃=20

X₄+X₉+X₁₄+X₁₉+X₂₄=1

**3.1 Библиотека scipy.optimize**

from scipy.optimize import linprog # подключение библиотеки для работы скрипта

import time # подключение библиотеки для работы таймера

start = time.time() # старт таймера

c = [10, 26, 39, 7, 27,

19, 11, 24, 8, 12,

44, 14, 12, 33, 15,

36, 33, 7, 25, 5,

4, 32, 13, 23, 3] # коэффициенты целевой функции

b\_ub = [40,15,7,10,8] # правая часть ограничений – верхних границ

A\_ub = [[1,1,1,1,1, 0,0,0,0,0, 0,0,0,0,0 ,0,0,0,0,0 ,0,0,0,0,0],

[0,0,0,0,0, 1,1,1,1,1, 0,0,0,0,0 ,0,0,0,0,0 ,0,0,0,0,0],

[0,0,0,0,0, 0,0,0,0,0, 1,1,1,1,1 ,0,0,0,0,0 ,0,0,0,0,0],

[0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0 ,1,1,1,1,1 ,0,0,0,0,0],

[0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0 ,0,0,0,0,0 ,1,1,1,1,1]] #двумерный массив ndarray (или список списков) с коэффициентами ограничений –верхних границ («≤»);

b\_eq = [10,8,40,20,1] # правая часть ограничений равенства;

A\_eq = [[1,0,0,0,0, 1,0,0,0,0, 1,0,0,0,0, 1,0,0,0,0, 1,0,0,0,0 ],

[0,1,0,0,0, 0,1,0,0,0, 0,1,0,0,0, 0,1,0,0,0, 0,1,0,0,0 ],

[0,0,1,0,0, 0,0,1,0,0, 0,0,1,0,0, 0,0,1,0,0, 0,0,1,0,0],

[0,0,0,1,0, 0,0,0,1,0, 0,0,0,1,0, 0,0,0,1,0, 0,0,0,1,0],

[0,0,0,0,1, 0,0,0,0,1, 0,0,0,0,1, 0,0,0,0,1, 0,0,0,0,1]] # двумерный массив ndarray (или список списков) с коэффициентами ограничений равенства;

print(linprog(c, A\_ub, b\_ub, A\_eq, b\_eq))

stop = time.time()

print ("Время :")

print(stop - start)

Выполнение данного кода приведёт к выводу функцией print текстового представления всех полей объекта scipy.optimizeResult, возращенного linprog:

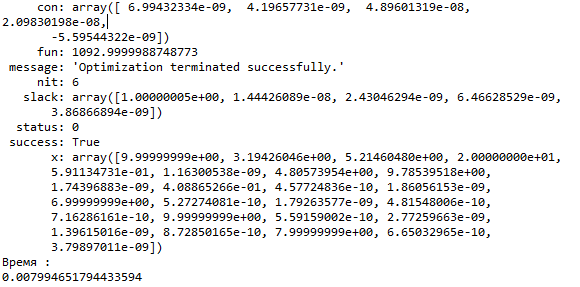


Рисунок 2. Screenshot выполнения кода в программе spyder. Где nit=6-это кол-во итераций, fun: 1092.9999988748773 –значение целевой ф-и. status: 0- статус решения: 0- поиск оптимального решения завершился успешно.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 9.99999999e+00 | 3.19426046e+00 | 5.21460480e+00 | 2.00000000e+01 | 5.91134731e-01 |
| 1.16300538e-09 | 4.80573954e+00 | 9.78539518e+00 | 1.74396883e-09 | 4.08865266e-01 |
| 4.57724836e-10 | 1.86056153e-09 | 6.99999999e+00 | 5.27274081e-10 | 1.79263577e-09 |
| 4.81548006e-10 | 7.16286161e-10 | 9.99999999e+00 | 5.59159002e-10 | 2.77259663e-09 |
| 1.39615016e-09 | 8.72850165e-10 | 7.99999999e+00 | 6.65032965e-10 | 3.79897011e-09 |

Таблица 2. Результат работы кода из библиотеки scipy.optimize сведён в таблицу.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Сколько нужно | B1=10 | B2=8 | B3=40 | B4=20 | B5=1 |
|  | 10\*10 | 3\*26 | 5\*39 | 20\*7 | 1\*27 |
| 0 | 5\*11 | 10\*24 | 0 | 0 |
| 0 | 0 | 7\*12 | 0 | 0 |
| 0 | 0 | 10\*7 | 0 | 0 |
| 0 | 0 | 8\*13 | 0 | 0 |
| Сколько доставили | 10 | 8 | 40 | 20 | 1 |

Таблица 3. Округление результата, представленного в таблице 2 умноженные на коэффициенты из таблицы 1.

Далее произведём расчёт стоимости доставки всех товаров потребителями с помощью алгоритма из библиотеки scipy.optimize:

OptimalX=100+78+55+195+240+84+70+104+140+27=1093.

**3.2 Библиотека Cvxopt.modeling.**

from cvxopt.modeling import variable, op # подключение библиотеки для работы скрипта

import time # подключение библиотеки для работы таймера

start = time.time() # старт таймера

x = variable(25, 'x')

c= [10, 26, 39, 7, 27, 19, 11, 24, 8, 12, 44, 14, 12, 33, 15, 36, 33, 7, 25, 5, 34, 32, 13, 23, 3] # коэффициенты целевой функции

z=(c[0]\*x[0] + c[1]\*x[1] + c[2]\*x[2] + c[3]\*x[3] + c[4]\*x[4] + c[5]\*x[5] + c[6]\*x[6] + c[7]\*x[7] + c[8]\*x[8] + c[9]\*x[9] + c[10]\*x[10] + c[11]\*x[11] + c[12]\*x[12] + c[13]\*x[13] + c[14]\*x[14] + c[15]\*x[15] + c[16]\*x[16] + c[17]\*x[17] + c[18]\*x[18] + c[19]\*x[19] + c[20]\*x[20] + c[21]\*x[21] + c[22]\*x[22] + c[23]\*x[23] + c[24]\*x[24])

mass1 = (x[0] + x[1] + x[2] + x[3] + x[4] <= 40)

mass2 = (x[5] + x[6] + x[7] + x[8] + x[9] <= 15)

mass3 = (x[10] + x[11] + x[12] + x[13] + x[14] <= 7)

mass4 = (x[15] + x[16] + x[17] + x[18] + x[19] <= 10)

mass5 = (x[20] + x[21] + x[22] + x[23] + x[24] <= 8)

mass6 = (x[0] + x[5] + x[10] + x[15] + x[20] == 10)

mass7 = (x[1] + x[6] + x[11] + x[16] + x[21] == 8)

mass8 = (x[2] + x[7] + x[12] + x[17] + x[22] == 40)

mass9 = (x[3] + x[8] + x[13] + x[18] + x[23] == 20)

mass10 = (x[4] + x[9] + x[14] + x[19] + x[24] == 1)#ограничение mass1-10

x\_non\_negative = (x >= 0) # так как путь не мб отрицательным

problem =op(z,[mass1,mass2,mass3,mass4 ,mass5,mass6, mass7,mass8 ,mass9, mass10,x\_non\_negative])

problem.solve(solver='glpk')

print("Результат Xopt:")

for i in x.value:

print(i)

print("Стоимость доставки:")

print(problem.objective.value()[0])

stop = time.time()

print ("Время :")

print(stop - start)

результат выполнения скрипта:

*Результат Xopt:*

*10.0 8.0 0.0 20.0 1.0 0.0 0.0 15.0 0.0 0.0 0.0 0.0 7.0 0.0 0.0 0.0 0.0 10.0 0.0 0.0 0.0 0.0 8.0 0.0 0.0 Стоимость доставки:*

*1093.0*

*Время :*

*0.006994724273681641*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 8 | 0 | 20 | 1 |
| 0 | 0 | 15 | 0 | 0 |
| 0 | 0 | 7 | 0 | 0 |
| 0 | 0 | 10 | 0 | 0 |
| 0 | 0 | 8 | 0 | 0 |

Таблица 4. Данные работы кода из библиотеки cvxopt.modeling сведены в таблицу.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Сколько нужно | B1=10 | B2=8 | B3=40 | B4=20 | B5=1 |
|  | 10\*10 | 8\*26 | 0 | 20\*7 | 1\*27 |
| 0 | 0 | 15\*24 | 0 | 0 |
| 0 | 0 | 7\*12 | 0 | 0 |
| 0 | 0 | 10\*7 | 0 | 0 |
| 0 | 0 | 8\*13 | 0 | 0 |
| Сколько доставили | 10 | 8 | 40 | 20 | 1 |

Таблица 5. Данные из таблицы 4 умноженные на данные из таблицы 1 и сведены в таблицу.

optimalX=100+208+360+84+70+104+140+27=1093.

**Библиотека pulp.**

from pulp import \*

#"подключаем библиотеку"

#conda install -c conda-forge pulp прописать в anaconda Prompt от имени администратора

# если не устанавливать, то anaconda пишет error pulp

import time # нужно для расчёта времи выполнения программы

start = time.time() # старт счётчика времени

x0 = pulp.LpVariable("x0", lowBound=0) # думал как через цикл заполнить так и не предумал

x1 = pulp.LpVariable("x1", lowBound=0)

x2 = pulp.LpVariable("x2", lowBound=0)

x3 = pulp.LpVariable("x3", lowBound=0)

x4 = pulp.LpVariable("x4", lowBound=0)

x5 = pulp.LpVariable("x5", lowBound=0)

x6 = pulp.LpVariable("x6", lowBound=0)

x7 = pulp.LpVariable("x7", lowBound=0)

x8 = pulp.LpVariable("x8", lowBound=0)

x9 = pulp.LpVariable("x9", lowBound=0)

x10 = pulp.LpVariable("x10", lowBound=0)

x11 = pulp.LpVariable("x11", lowBound=0)

x12 = pulp.LpVariable("x12", lowBound=0)

x13 = pulp.LpVariable("x13", lowBound=0)

x14 = pulp.LpVariable("x14", lowBound=0)

x15 = pulp.LpVariable("x15", lowBound=0)

x16 = pulp.LpVariable("x16", lowBound=0)

x17 = pulp.LpVariable("x17", lowBound=0)

x18 = pulp.LpVariable("x18", lowBound=0)

x19 = pulp.LpVariable("x19", lowBound=0)

x20 = pulp.LpVariable("x20", lowBound=0)

x21 = pulp.LpVariable("x21", lowBound=0)

x22 = pulp.LpVariable("x22", lowBound=0)

x23 = pulp.LpVariable("x23", lowBound=0)

x24 = pulp.LpVariable("x24", lowBound=0)

costWay = [10, 26, 39, 7, 27,

19, 11, 24, 8, 12,

44, 14, 12, 33, 15,

36, 33, 7, 25, 5,

34, 32, 13, 23, 3]# стоимость перевозки 1-го комплекта заказа по маршруту аn(склада) до bn (потребителя)

problem = pulp.LpProblem('0',pulp.LpMaximize)

problem += -costWay[0]\*x0- costWay[1]\*x1-costWay[2]\*x2-costWay[3]\*x3-costWay[4]\*x4- costWay[5]\*x5- costWay[6]\*x6-costWay[7]\*x7-costWay[8]\*x8-costWay[9]\*x9-costWay[10]\*x10- costWay[11]\*x11-costWay[12]\*x12-costWay[13]\*x13-costWay[14]\*x14- costWay[15]\*x15- costWay[16]\*x16-costWay[17]\*x17-costWay[18]\*x18-costWay[19]\*x19- costWay[20]\*x20- costWay[21]\*x21-costWay[22]\*x22-costWay[23]\*x23-costWay[24]\*x24, "Функция цели"

problem +=x0 + x1 +x2 + x3 + x4 <= 40,"1"

problem +=x5 + x6 +x7 + x8 + x9 <= 15, "2"

problem +=x10 + x11 +x12 + x13 + x14 <= 7, "3"

problem +=x15 + x16 +x17 + x18 + x19 <= 10, "4"

problem +=x20 + x21 +x22 + x23 + x24 <= 8, "5"

problem +=x0+ x5+ x10 + x15 + x20 == 10, "6"

problem +=x1+ x6+ x11 + x16 + x21 == 8, "7"

problem +=x2+ x7+ x12 + x17 + x22 == 40, "8"

problem +=x3+ x8+ x13 + x18 + x23 == 20, "9"

problem +=x4+ x9+ x14 + x19 + x24 == 1, "10"

problem.solve()

print ("Результат:")

for variable in problem.variables():

print (variable.name, "=", variable.varValue)

print ("Стоимость доставки:")

print (abs(value(problem.objective)))

stop = time.time()

print ("Время :")

print(stop - start)

результат:

x0 = 10.0 x1 =8.0 x10=0.0 x11 = 0.0 x12 = 7.0 x13 = 0.0 x14 = 0.0 x15 = 0.0 x16 = 0.0 x17 = 10.0

X18 = 0.0 x19 = 0.0 x2 = 0.0 x20 = 0.0 x21 = 0.0 x22 = 8.0 x23 = 0.0 x24 = 0.0 x3 = 20.0 x4 = 10

X5 = 0.0 x6 =0.0 x7 = 15.0 x8 = 0.0 x9 = 0.0

Стоимость доставки:

1093,0

Время:

0.045976877212524414

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 8 | 0 | 20 | 1 |
| 0 | 0 | 15 | 0 | 0 |
| 0 | 0 | 7 | 0 | 0 |
| 0 | 0 | 10 | 0 | 0 |
| 0 | 0 | 8 | 0 | 0 |

Таблица 6. Данные работы кода из библиотеки pulp сведены в таблицу.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Сколько нужно | B1=10 | B2=8 | B3=40 | B4=20 | B5=1 |
|  | 10\*10 | 8\*26 | 0 | 20\*7 | 1\*27 |
| 0 | 0 | 15\*24 | 0 | 0 |
| 0 | 0 | 7\*12 | 0 | 0 |
| 0 | 0 | 10\*7 | 0 | 0 |
| 0 | 0 | 8\*13 | 0 | 0 |
| Сколько доставили | 10 | 8 | 40 | 20 | 1 |

Таблица 7. Данные из таблицы 7 умноженные на данные из таблицы 1 и сведены в таблицу.

optimalX=100+208+360+84+70+104+140+27=1093.

**Сравнение библиотек и вывод.**

|  |  |  |  |
| --- | --- | --- | --- |
| Категория | scipy.optimize | *Cvxopt.modeling* | pulp |
| Время выполнения | 0.00799417495727539 | *0.006994724273681641* | 0.045976877212524414 |

Таблица 8. Сравнение методов из 3-х библиотек.

В лабораторной работе были рассмотрены 3 библиотеки для решения закрытой транспортной задачи. Как показывает таблица 8. Скорость выполения cvxopt.modeling самая высокая для решения данной задачи. Для работы 3-х библиотек необходимы сторонние библиотеки, которые приходится скачивать и устанавливать отдельно, что является минусом. При работе с кодом была изучена литература [1-5], как англоязычная, так и русскоязычная. Среди литературы можно выявить «фаворита»-это библиотека scipy.optimize. Библиотека scipy.optimize позволила получить другой маршрут транспортировки товара от склада до заказчика при равных затратах на транспортировку.

Список источников:

1.Решение задач линейного программирования с использованием GNU Octave, GLPK и Python А.В. Пономарев <https://cais.iias.spb.su/ponomarev/LP_tutorial.pdf>

2. <https://habr.com/ru/post/335104/>

3. <https://scaron.info/blog/linear-programming-in-python-with-cvxopt.html>

4. <https://pythonhosted.org/PuLP/>

5. <https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html>