MP rep1　　205713B 朝比奈太郎　2022/05/07

(1)

ナップサック問題 => キャリーケース問題

キャリーケース問題

某航空会社のプランでは、機内持ち込み可能な手荷物を 7kg 以内かつ三辺の合計値が 115cm 以内と決められている。この条件を活かして、7kg で体積 56000cm(40cm, 40cm, 35cm とした場合)で一番効率よく機内持ち込み手荷物にするにはどうすればよいか。

<定式化>

キャリーケースの容量を B, 全部で N 個の荷物, 荷物 I の体積を Ai, 重量を Ci, 荷物 i をキャリーケースに入れることを Xi=1, 入れないことを Xi=0 とすると、

max Σ(i=1 to N)Ci･Xi

subject to Σ(i=1 to N)Ai･Xi≦B , Xi∈{0,1} (i=1,･･･,N)

輸送問題 => 新聞配達問題

新聞配達問題

某新聞会社では 2 種類の新聞するための工場をそれぞれ 1 個ずつ計 2 つ保有しており、6 つの支店に新聞を輸送している。各工場の保有量を ai, 支店 j での 1 日需要を bj, 工場 i から支店 j にかかる輸送費を 1 種類あたり CIj とする。1 日の総輸送費を最小にするように各工場から各支店への輸送量 Xij を求めよ。

<定式化>

Min Z = Σ(i=1 to 2)　Σ(i=1 to 6)Cij･Xij

Subject to　Σ(j=1 to 6)Xij ≦ ai, i=1,2

Σ(i=1 to 3)Xij ≧ bi, j=1,2,3,4,5

Xij ≧ 0, (i=1,2　j=1,2,3,4,5,6)

(2)

ガウスジョルダン法を採択。連立方程式を行列に変化させて解く方法である。

```
import sys
import traceback

class GaussJorden:
    def __init__(self):
        self.a = [
            [ 5, -4,  6,  8],
            [ 7,  -6, 10,  14],
```

```
            [ 4,   9, 7,   74]
        ]
        self.n = len(self.a)

    def exec(self):
        """ Solving and display """
        try:
            self.__display_equations()
            for k in range(self.n):
                p = self.a[k][k]
                for j in range(k, self.n + 1):
                    self.a[k][j] /= p
                for i in range(self.n):
                    if i == k:
                        continue
                    d = self.a[i][k]
                    for j in range(k, self.n + 1):
                        self.a[i][j] -= d * self.a[k][j]
            self.__display_answers()
        except Exception as e:
            raise

    def __display_equations(self):
        """ Display of source equations """
        try:
            for i in range(self.n):
                for j in range(self.n):
                    print("{:+d}x{:d} ".format(self.a[i][j], j + 1), end="")
                print("= {:+d}".format(self.a[i][self.n]))
        except Exception as e:
            raise

    def __display_answers(self):
        """ Display of answer """
        try:
            for k in range(self.n):
```

```
                    print("x{:d} = {:f}".format(k + 1, self.a[k][self.n]))
            except Exception as e:
                raise

if __name__ == '__main__':
    try:
        obj = GaussJorden()
        obj.exec()
    except Exception as e:
        traceback.print_exc()
        sys.exit(1)
```

実行結果

```
(base) asahinatarou@talol 1 % /opt/miniconda3/bin/python /Users/taro/MP/1/Gjordan.py
+5x1 -4x2 +6x3 = +8
+7x1 -6x2 +10x3 = +14
+4x1 +9x2 +7x3 = +74
x1 = 2.000000
x2 = 5.000000
x3 = 3.000000
```

(3)総当たり解法プログラム

【α1】

荷物i	1	2	3	4	5	6	7	8	B
重量Ai	3	6	5	4	8	5	3	4	25
価格Ci	7	12	9	7	13	8	4	5	

```
import time


start = time.time()
# 物の個数
n = 8
# 要領
capacity = 25
#物の重さ、価値
size = [3, 6, 5, 4, 8, 5, 3, 4]
price = [7, 12, 9, 7, 13, 8, 4, 5]
#最高の重さと価格と最適な組み合わせを記録する
max_size = -1
```

```
max_price = -1
combination = []
# i には 2**n までの値が入る。
for i in range(2 ** n):
    #  変数の初期化
    tmp_size = 0
    tmp_price = 0
    tmp_combination = []
    over_flag = False
    for j in range(n):
        # 2 進数で計算。シフトして 1 ビットずつ判断。
        is_put = i >> (n - j - 1) & 1
        #  値を入力
        tmp_combination.append(is_put)
        tmp_size += is_put * size[j]
        tmp_price += is_put * price[j]
        #print(tmp_combination)
        # capa  を越えたらフラグを立てて  break
        if tmp_size > capacity:
            over_flag = True
            break
    # over flag  が立ってない  かつ  暫定  max price  より高いときに更新
    if (not over_flag) and tmp_price > max_price:
        max_price = tmp_price
        max_size = tmp_size
        combination = tmp_combination
print("合計が最大になる組み合わせ")
print(combination)
print("合計価格: ", max_price)
print("合計サイズ: ", max_size)

end = time.time() - start
print(f"{end:.3f}s")
```

実行結果

```
(base) asahinatarou@talol 1 % /opt/miniconda3/bin/python /Users/taro/MP/1/try.py
合計が最大になる組み合わせ
[1, 1, 1, 0, 1, 0, 1, 0]
合計価格：  45
合計サイズ：  25
0.002s
```

(4)beta1 のナップサック問題総当たり解法

【β1】

荷物 i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	B
重量 Ai	3	6	5	4	8	5	3	4	3	5	6	4	8	7	11	8	14	6	12	4	55
価格 Ci	7	12	9	7	13	8	4	5	3	10	7	5	6	14	5	9	6	12	5	9	

(3)と同じプログラムで重量と価格、荷物の数をそれぞれ beta1 の問題に対応させる。
プログラムは(3)と同じ
実行結果

```
(base) asahinatarou@talol 1 % /opt/miniconda3/bin/python /Users/taro/MP/1/try.py
合計が最大になる組み合わせ
[1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
合計価格：  102
合計サイズ：  55
15.157s
```

(5)高速化の方法、その際の実行時間

```
(base) asahinatarou@talol 1 % /Users/taro/.venv/dm/bin/python /Users/taro/MP/1/try.py
合計が最大になる組み合わせ
[1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
合計価格：  102
合計サイズ：  55
10.500s
```

(4)までは、自分の mac にインストールしている、python3.7.6 を使用していたが、高速化を図るために仮想環境を構築し、そこに python3.9.12 64bit をインストールした。その環境で実行すると、3.7.6 を使用した時と比べて約 4.5s 短縮することができ、10.5s となった。