数理計画とアルゴリズム rep2　　205713B 朝比奈太郎 2022/05/25

【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	

【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	

(1) 欲張り法でα1，β1をそれぞれ解いてみよ。

α1

```
num = 8
x_num = []
x_num = []
x_weight = [3, 6, 5, 4, 8, 5, 3, 4]
x_price = [7, 12, 9, 7, 13, 8, 4, 5]
x_value = [] #品物の価値  price / weight
x_reValue = []
max = 25
storage = 0
price = 0

for i in range(num):
    value = x_price[i] / x_weight[i]
    x_value.append(value)

for j in range(num):
    if storage + x_weight[j] <= max:
        storage += x_weight[j]
```

```
            price += x_price[j]
            x_num.append(1)
        else:
            x_num.append(0)

print("荷物の組み合わせは")
print(x_num)
print("価格は: " + str(price))
print("最大容量は: " + str(storage))
```

実行結果

```
(base) asahinatarou@talol MP % /opt/miniconda3/bin/python /Users/taro/MP/2/greedy.py
荷物の組み合わせは
[1, 1, 1, 1, 0, 1, 0, 0]
価格は: 43
最大容量は: 23
```

β 1

```
num = 20
x_num = []
x_weight = [3, 6, 5, 4, 8, 5, 3, 4, 3, 5, 6, 4, 8, 7, 11, 8, 14, 6, 12, 4]
x_price = [7, 12, 9, 7, 13, 8, 4, 5, 3, 10, 7, 5, 6, 14, 5, 9, 6, 12, 5, 9]
x_value = []   # 品物の価値  price / weight
x_reValue = []
max = 55
storage = 0
price = 0

#tekitou = []
#change_num = []
for i in range(num):
    value = x_price[i] / x_weight[i]
    x_value.append(value)
#print(x_value)

#価値の高い順に並び替える => 価値の高い順からバッグに入れていく
```

```
x_reValue = sorted(x_value, reverse=True)
#print(x_reValue)
"""
for i in range(num):
    for j in range(num):
        if x_reValue[i] == x_price[j] / x_weight[j]:
            if not i in tekitou:
                print(i+1, j+1)
                tekitou.append(i)
                change_num.append(j)

print(change_num) #価値の高い順番を示した list => 失敗？
"""
#x_value と x_reValue を比べて手動でソートする。ここのプログラム実装ができませんでした。
X_weight = [3, 4, 6, 5, 7, 6, 5, 4, 8, 5, 3, 4, 4, 6, 8, 3, 8, 11, 14, 12]
X_price = [7, 9, 12, 10, 14, 12, 9, 7, 13, 8, 4, 5, 5, 7, 9, 3, 6, 5, 6, 5]
for k in range(num):
    if storage + X_weight[k] <= max:
        storage += X_weight[k]
        price += X_price[k]
        x_num.append(1)
    else:
        x_num.append(0)

print("荷物の組み合わせ(ソート後)は")
print(x_num)
print("価格は: " + str(price))
print("最大容量は: " + str(storage))
```

実行結果

```
(base) asahinatarou@talol MP % /opt/miniconda3/bin/python /Users/taro/MP/2/try.py
荷物の組み合わせ(ソート後)は
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
価格は: 101
最大容量は: 53
```

(2) 上記の結果に考察を加えるとともに、最適解(近似解)を効率よく求めるための工夫を考えて検証せよ。

上記のプログラムは、荷物が価値の高い順に並んでいた場合に非常に有効である。特に $\alpha 1$ のプログラムにおいて、荷物が価値の高い順番にソート済みであることを想定して作成したプログラムであるため無駄がないと言える。しかし、$\beta 1$ においてはソート済みではない(荷物が価値の高い順番には並んでいない)ため、それぞれの荷物の価値を計算し、計算結果を降順にソートした。計算結果を降順にソートしその順番を荷物にも適用させるプログラムの設計がうまくいかず、手動になってしまった。この箇所を訂正し、実装することができれば全自動でできるので短時間高精度に問題を解くことができると考える。

今回の $\alpha 1$, $\beta 1$ の問題は前回の全探索で最適解がもとまっているため、欲張り法の精度がわかる。私は、荷物数(サンプル数)が少ない方が最適解に近い答えが出るのではないかと考えた。そこで $\alpha 1, \beta 1$ の最適解と今回作成した欲張りほうでもとまった答えを手足し合わせて検証した。

$\alpha 1$ の場合

```
(base) asahinatarou@talol MP % /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
0.001s
(base) asahinatarou@talol MP % /opt/miniconda3/bin/python /Users/taro/MP/2/greedy.py
荷物の組み合わせは
[1, 1, 1, 1, 0, 1, 0, 0]
価格は: 43
最大容量は: 23
```

$\beta 1$ の場合

```
(base) asahinatarou@talol MP % /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
11.983s
(base) asahinatarou@talol MP % /opt/miniconda3/bin/python /Users/taro/MP/2/try.py
荷物の組み合わせ(ソート後)は
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
価格は: 101
最大容量は: 53
```

検証結果、予想とは裏腹にサンプル数が多い $\beta 1$ の方が最適解 102 に対して欲張り法でもとまった解が 101 であるという最適解に近い結果となった。

欲張り法は必ずしも最適解がもとまるとは限らない。しかし、単純で高速であることが多いため他の解法と組み合わせることでより有効なアルゴリズムになりえる。また、最小スパニング木問題や最短経路問題においては最適解がもとまる。

従って、最適解を効率よく求めるための工夫は、ます欲張り法のみで最適解がもとまるプログラムであるかどうかを判断し、欲張り法のみでは最適解がもとまらない問題である場合は他の解法と欲張り法を組み合わせることであると考える。