数理計画とアルゴリズム
レポート3

175751C
宮城孝明

1)欲張り法の例題(LSI設計問題)に対して、吝嗇法を適用した解を求めよ

吝嗇法でLSI問題を解く。
まずはじめに、全体の面積の和を求める。 合計値は34mm^2となる
そして、各チップの価値を求めていく。2.33,2,1.8,1.75,1.625,1.6,1.33,1.25となる
ここのままでは、条件が成り立たないため、値が小さい(価値が低い)順から合計値の面積から取り除く
そして、合計の面積は18mm^2となる。その時の性能は、35となる

2)欲張り法と吝嗇法で解が一致するか？あるいは、両者で解はどのように異なるか？について考察せよ

今回の問題では、解は一致しなかった。
理由としては、欲張り法では良い性能を持つものから順に入れていくという手法であるから、結果が良くなりそうな値が増えていくため、解の近似値を求めることができた
一方、吝嗇法は条件を満たさないといけないため、要素を常に削除し続け、値を取り入れることをしない。そのため、結果が大幅に変わってしまった。

3)欲張り法のプログラムを作成し、前回のレポート課題の問題α1とβ1をそれぞれ解け
[ソースコード]

```
import os
import sys
from collections import defaultdict, Counter
from itertools import product, permutations,combinations, accumulate
from operator import itemgetter
from bisect import bisect_left,bisect
from heapq import heappop,heappush,heapify
from math import ceil, floor, sqrt, gcd
from copy import deepcopy
import numpy as np


def main():


   size = [3, 6, 5, 4, 8, 5, 3, 4]
```

```
    value = [7, 12, 9, 7, 13, 8, 4, 5]
    area = 0
    """
    size = [3, 6, 5, 4, 8, 5, 3, 4, 3, 5, 6, 4, 8, 7, 11, 8, 14, 6, 12, 4]
    value = [7, 12, 9, 7, 13, 8, 4, 5, 3, 10, 7, 5, 6, 14, 5, 9, 6, 12, 5, 9]
    """
    result_list = []
    ans = [0]*len(size)
    for i in range(len(size)):
        num = size[i]
        val = value[i]
        result_list.append(val/num)

    result = []
    for i in range(len(result_list)):
        result.append([result_list[i], size[i]])

    result.sort(reverse=True)
    for i in range(len(result)):
        area += result[i][1]
        if area <= 25:
            ans[i] = 1
        else:
            area -= result[i][1]
    print(ans)
if __name__ == "__main__":
    main()
```

[結果]

α1

[1, 1, 1, 1, 0, 1, 0, 0] 43 23

β1

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] 101 53

4)上記(3)の結果について考察を加えるとともに、最適解(またはできるだけ良い近似解)を効率良く求めるための工夫を考えて検証せよ

この結果を見るからには、適切な最適解をとることは難しいと考えられる。適切な最適解に、常に良い値(価値が高い値)が答えの中に含まれていると限らない。そのため、総当

たりでもしないかぎり、適当な最適解を見つけることはできない。
 吝嗇法の改良して、最適解または近似解が取れるではないかと考える。どのように改良するかというと、価値の低い値を抜き取る処理を行った後に、抜き取った値から条件に当てはまる値を挿入する処理を入れる。これにより、入ることができる値を余すことなく利用することができ、より最適解に近似した値を取得可能だと考えられる。

[ソースコード]

```
import os
import sys
from collections import defaultdict, Counter
from itertools import product, permutations,combinations, accumulate
from operator import itemgetter
from bisect import bisect_left,bisect
from heapq import heappop,heappush,heapify
from math import ceil, floor, sqrt, gcd
from copy import deepcopy
import numpy as np

def main():

   size = [3, 6, 5, 4, 8, 5, 3, 4]
   value = [7, 12, 9, 7, 13, 8, 4, 5]
   #size = [3, 6, 5, 4, 8, 5, 3, 4, 3, 5, 6, 4, 8, 7, 11, 8, 14, 6, 12, 4]
   #value = [7, 12, 9, 7, 13, 8, 4, 5, 3, 10, 7, 5, 6, 14, 5, 9, 6, 12, 5, 9]

   area = 0
   #limit_num = 55
   limit_num = 25
   result_list = []
   ans = [1]*len(size)
   ans_num = 0
   max_area = 0

   for i in range(len(size)):
       num = size[i]
       val = value[i]
       result_list.append(val/num)

   result = []
   for i in range(len(result_list)):
       result.append([result_list[i], size[i], value[i]])
```

```
    for i in range(len(result)):
        max_area += result[i][1]
        ans_num += result[i][2]
    result.sort()

    del_list = []
    for i in range(len(result)):
        max_area -= result[i][1]
        ans_num -= result[i][2]
        del_list.append([result[i][1], result[i][2]])
        ans[i] = 0
        if limit_num < max_area:
            pass
        else:
            break
    ans.sort(reverse=True)
    print(ans, ans_num, max_area)

    del_list.sort(reverse=True)
    for i in range(len(del_list)):
        if max_area + del_list[i][0] <= limit_num:
            ans[len(ans) - len(del_list) + i] = 1
            max_area += del_list[i][0]
            ans_num += del_list[i][1]
        else:
            pass

    print(ans, ans_num, max_area)

if __name__ == "__main__":
    main()
```

[結果]

```
[1, 1, 1, 1, 0, 0, 0, 0] 35 18      # 改善前
[1, 1, 1, 1, 0, 1, 0, 0] 43 23      # 改善後
```

[考察]

結果しょふ吝嗇法の改善後の処理では近似値が最適値により近づいていることがわかる。しかし、これはあくまでも欲張り法と変わらない結果となっている。そのことから、最適値をとるためにはまだ別の手法があるのではないかと考えられる。