Universidade Federal do Mato Grosso do Sul
Campus Ponta Porã
Teoria da computação

Implementação das heuristicas Gulosa e GRASP para o Problema da Mochila

Aluno: Beatriz Camargo Câmara

Professor: Eduardo Teodoro Bogue

Outubro 2019

Introdução

Este trabalho consiste na construção de uma heurística gulosa e uma heurística GRASP para o problema da mochila. O problema da mochila consiste em um problema de otimização combinatória a qual é necessário colocar objetos diferentes em uma mochila que possuem valores e pesos específicos de maneira que possa-se obter o maior lucro possível sem ultrapassar o peso máximo da mochila.

Implementação Gulosa

Na implementação da heuristica gulosa foi utilizado como critério os items com maior razão valor-peso que se mantivessem dentro da capacidade da mochila. Foi criado um vetor de tuplas chamado *item* que possui a lista organizada de maneira decrescente pela razão. Logo é feita a verificação, caso o peso do item seja menor que a capacidade da mochila é adcionado o lucro do item a mochila e decrementado a capacidade da mochila, até que acabem os items ou a capacidade seja igual a zero.

```
1  def media(value, weight):
2      item = {}
3      for i in range(len(value)):
4          item[i] = np.divide(value[i], weight[i]), value[
               i], weight[i], i
5
6      item = sorted(item.values(), reverse=True)
7
8      return item
9
10  def greedy(value, weight, capacity):
11     item = media(value, weight)
12     value_max = 0
13     for i in range(len(value)):
14         if capacity == 0:
15             break
16         if item[i][2] <= capacity:
17             value_max += item[i][1]
18             capacity -= item[i][2]
19     print(value_max)
```

Implementação GRASP

Na implementação da heurística GRASP, a fase de Construção é feita através da função $greedy_randomized_contruction$ que recebe a lista dos itens(que estão ordenados pela razão assim como na implementação gulosa) e a capacidade da mochila. O k é definido como 2, que é o comprimento do laço for o qual insere os valores e atualiza o LCR. Em seguida escolhe-se entre os valores do LCR qual utilizar de maneira aleatória e armazena-o na variável $random_item$ é verificado se o item escolhido cabe na mochila, caso positivo o vetor $solution_temp$, que serve para guardar as soluções temporárias, recebe o índice desse item, a capacidade é decrementada e é retirado da cópia da lista o $random_item$, isto é feito para cada um dos itens. Ao final de todas as iterações é criado um vetor $solution$ que receberá 0 nas posições dos itens que não foram inseridos na mochila 1 para os que foram.
A fase da Busca Local localiza-se na função $local_search$ que recebe o $solution$ do $greedy_randomized_contruction$, a lista de itens, a capacidade da mochila, a lista de lucros e pesos . É criado uma variável que recebe o lucro da solução atual chamada de $currentProfit$, o qual é calculado na função $profit_calculate$. Após isso é mudado um dos bits da solução atual para obter a solução vizinha e é calculado o seu lucro, caso o lucro da solução vizinha seja maior que a da $currentProfit$, então atualiza-o e o $best_solution$ (melhor solução) passa a ser a vizinha. A solução volta a inicial e reinicia-se o ciclo até percorrer todos os vizinhos possíveis, e assim que finalizar retorna a $best_solution$. A janela de execução é de 50.

```
1  def media(value, weight):
2      item = {}
3      for i in range(len(value)):
4          item[i] = np.divide(value[i], weight[i]), value[
               i], weight[i], i
5
6      item = sorted(item.values(), reverse=True)
7
8      return item
9
10
11 def profit_calculate(solution, item, capacity, value,
       weight):
12     profit = 0
13     temp_item = {}
14     for i in range(len(value)):
```

```
        temp_item[i] = value[i], weight[i]
    for i in range(len(solution)):
        if solution[i] == 1:
            weight = temp_item[i][1]
            capacity -= weight
            if(capacity < 0):
                return -1
            else:
                profit += temp_item[i][0]
    return profit


def local_search(solution, item, capacity, value, weight
    ):
    best_solution = solution[:]
    begin_solution = solution[:]
    currentProfit = profit_calculate(
        begin_solution, item, capacity, value, weight)
    for i in range(len(solution)):
        if solution[i] == 1:
            solution[i] = 0
        else:
            solution[i] = 1

        neighborProfit = profit_calculate(
            solution, item, capacity, value, weight)

        if neighborProfit > currentProfit:
            currentProfit = neighborProfit
            best_solution = solution[:]

        if solution[i] == 1:
            solution[i] = 0
        else:
            solution[i] = 1
    return best_solution


def greedy_randomized_contruction(item, capacity):
    temp = item[:]
    solution_temp = []
    while len(temp) > 0:
        LCR = []
```

```
57          for i in range(2):
58              if len(temp) > i:
59                  LCR.append(temp[i])
60          random_item = random.choice(LCR)
61
62          if random_item[2] <= capacity:
63              solution_temp.append(random_item[3])
64              capacity -= random_item[2]
65
66          temp.pop(temp.index(random_item))
67      solution = [0 for i in range(len(item))]
68      for i in solution_temp:
69          solution[i] = 1
70
71      return solution
72
73
74  def grasp(item, capacity, value, weight):
75      best_solution = 0
76      for i in range(50):
77          solution = greedy_randomized_contruction(item,
                capacity)
78          solution = local_search(solution, item, capacity
                , value, weight)
79          best_solution = max(
80              best_solution, profit_calculate(solution,
                    item, capacity, value, weight))
81
82      return best_solution
```

Tabela de Resultado

Segue a tabela com o resultado de todas as instâncias de entrada utilizadas.

	Grasp	Guloso
0	31621	29636
1	67829	64939
2	143449	143449
3	28840	28840
4	15785	15785
5	99861	99861
6	1922	1894
7	721	714
8	9787	9717
9	19274	17523
10	29965	29943
11	49885	49884
12	49398	49395
13	20880	20880
14	20676	20676
15	46281	46218