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

Implementação da Busca Tabu

Aluno: Beatriz Camargo Câmara

Professor: Eduardo Teodoro Bogue

Outubro 2019

Introdução

Este trabalho consiste na construção da busca Tabu para o problema da mochila que é um problema de otimização combinatória o 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. Neste trabalho é feita a comparação com os resultados da implementação das heuristicas Grasp e Gulosa feitas anteriormente.

Implementação da Busca Tabu

Na busca Tabu todos os processos foram feitos de forma semelhante a implementação do Grasp, as unicas partes que houveram alterações foi na $local_search$, onde ao invés de comparar todas as soluções para ver qual delas era melhor que a solução original, compara-se com a melhor entre os vizinhos. Além dessa alteração tem-se a função *tabu* que é semelhante a *grasp*, porém é feita uma lista T com as melhores soluções da janela de tempo 2. As entradas são executadas cada uma 100 vezes, exceto dos itens 11-15 pois devido ao tamanho das entradas foram executados apenas em 25 iterações.

```
1  import re
2  import random
3  import numpy as np
4  from operator import itemgetter
5
6
7  def media(value, weight):
8      item = {}
9      for i in range(len(value)):
10         item[i] = np.divide(value[i], weight[i]), value[
               i], weight[i], i
11
12     item = sorted(item.values(), reverse=True)
13
14     return item
15
16
17 def profit_calculate(solution, item, capacity, value,
       weight):
18     profit = 0
```

```
    temp_item = {}
    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[:]
    current_profit = profit_calculate(solution, item,
        capacity, value, weight)
    temp_profit = 0
    for i in range(len(solution)):
        if solution[i] == 1:
            solution[i] = 0
        else:
            solution[i] = 1

        temp_profit = profit_calculate(
            best_solution, item, capacity, value, weight
                )

        if(temp_profit >= current_profit):
            current_profit = temp_profit
            T = i
            final_solution = solution[:]
            final_profit = temp_profit

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

    return final_solution

```

```
59
60  def greedy_randomized_contruction(item, capacity):
61      temp = item[:]
62      solution_temp = []
63      while len(temp) > 0:
64          LCR = []
65          for i in range(2):
66              if len(temp) > i:
67                  LCR.append(temp[i])
68          random_item = random.choice(LCR)
69
70          if random_item[2] <= capacity:
71              solution_temp.append(random_item[3])
72              capacity -= random_item[2]
73
74          temp.pop(temp.index(random_item))
75      solution = [0 for i in range(len(item))]
76      for i in solution_temp:
77          solution[i] = 1
78
79      return solution
80
81
82  def tabu(item, capacity, value, weight):
83      profit = []
84      bt_max = 2
85      T = []
86      acc = 0
87      for i in range(100):
88          solution = greedy_randomized_contruction(item,
               capacity)
89          acc = 0
90          while acc <= bt_max:
91
92              solution = local_search(solution, item,
                   capacity, value, weight)
93              if(solution not in T):
94                  T.append(solution)
95                  profit.append(profit_calculate(
96                      solution, item, capacity, value,
                           weight))
97              acc = acc + 1
98          #temp = max(profit)
```

```
99          #ind = profit.index(temp)
100
101     return max(profit)
102
103
104 def main():
105     for it in range(16):
106         file = open("entradas/input"+str(it+1)+".in", "r
                ")
107         i = 1
108         n = None
109         capacity = None
110         item = []
111         value = []
112         weight = []
113         for line in file:
114             if i == 1:
115                 n = int(line)
116             elif 1 < i <= n+1:
117                 s = str(line)
118                 aux1, aux2, aux3 = s.split()
119                 item.append(int(aux1))
120                 value.append(int(aux2))
121                 weight.append(int(aux3))
122             else:
123                 capacity = int(line)
124             i += 1
125
126         # greedy(value, weight, capacity)
127         item = media(value, weight)
128         print(tabu(item, capacity, value, weight))
129
130
131 if __name__ == "__main__":
132     main()
```

Tabela de Resultado

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

	Grasp	Guloso	Tabu
0	31621	29636	31621
1	67829	64939	67829
2	143449	143449	143449
3	28840	28840	28840
4	15785	15785	15785
5	99861	99861	99861
6	1922	1894	99861
7	721	714	721
8	9787	9717	9798
9	19274	17523	19274
10	29965	29943	29965
11	49885	49884	49885
12	49398	49395	49398
13	20880	20880	20880
14	20676	20676	20676
15	46281	46218	46281