Universidade Federal de São Paulo

Instituto de Ciência e Tecnologia

Relatório 1
Bin Packing Problem

Beatriz Viana Ferreira 133548
Davi Juliano Ferreira Alves 133595

São José dos Campos
Dezembro, 2020

OBJETIVO

A proposta do relatório é modelar e desenvolver um método heurístico para resolver *Bin Packing Problem (BPP)*, inicialmente recebemos n objetos, tal que cada um terá um volume v_j, onde $j = 1, ..., n$ e um número ilimitado de recipientes iguais de capacidade inteira V. O objetivo é embalar todos os objetos em um número mínimo de recipiente para que o volume total embalado em qualquer recipiente não exceda a capacidade.

BIN PACKING PROBLEM (BPP)

VARIÁVEIS DE ENTRADA:

- Quantidade de objetos;
- Volume dos objetos;
- Volume dos recipientes.

OBJETIVO:

- Minimizar o número de recipientes

MODELAGEM DO PROBLEMA:

A modelagem do problema se dá de maneira quase intuitiva. Seja $y_i \in \{0, 1\}$, $V \in \mathbb{R}$ e com $i = \{1, ..., n\}$ os bins finitos e de mesmo volume V existentes para a formulação do problema e seja $x_{ij} \in \mathbb{R}$, com $i = \{1, ..., n\}$ e $j = \{1, ..., m\}$ os volumes que devem ser colocados dentro de cada um dos bins. Já definidas as variáveis de decisão e as variáveis de contorno, obteremos o seguinte modelo:

$$\text{MIN: } \sum_{i=1}^{n} y_i = Z.$$

$$\text{Sujeito à: } \sum_{i=1}^{m} x_{ij} \leq V y_i, \forall i \in \{1, ..., n\}$$

$$x_{11}, ..., x_{nm} \geq 0.$$

A função objetivo é dada pela função Z que é o somatório de valores que y_i pode obter, ou seja, se a mochila está ocupada ou não. O problema definido acima é um problema NP-Difícil, ou seja, a complexidade computacional do algorítmo que resolve por meio do modelo segue um trajeto não polinomial.

Definição 1. *Um problema H é dito NP-Completo quando ele pertence à classe NP e pode ser resolvido em tempo polinomial por uma máquina redutível de Turing.*

Definição 2. *Um problema H é dito NP-Difícil se, e somente se, existe um problema NP-Completo L que é Turing-redutível em tempo polinomial para H.*

De acordo com Miyazawa (2020), os problemas de empacotamento (Bin Packing Problem), como muitos de natureza combinatória, podem ser facilmente formulados e compreendidos, escondendo atrás de sua aparente simplicidade, a sua real complexidade. São problemas NP-difíceis não aproximáveis -em termos absolutos- além de certas

constantes, a menos que P=NP. Esse caráter complexo em termos de aproximabilidade absoluta, justifica o estudo desses problemas quanto à sua aproximabilidade em termos assintóticos. Faremos a comparação de duas ideias heurísticas, sendo elas:

- **IDEIA HEURÍSTICA 1:**

  1. Preencher o recipiente j até que não caiba mais nenhum objeto nele, logo passaremos para o recipiente $j+1$;
  2. Verificar se o objeto cabe, toda vez que couber subtrai o volume do recipiente pelo volume do objeto.
  3. Embaralha o vetor n vezes e salva a melhor solução.

- **IDEIA HEURÍSTICA 2:**

  1. Organizar o vetor de forma decrescente;
  2. Preencher o recipiente j até que não caiba mais nenhum objeto nele, logo passaremos para o recipiente $j+1$;
  3. Verificar se o objeto cabe, toda vez que couber subtrai o volume do recipiente pelo volume do objeto.

```
1  import random
2  import time
3
4  def func_minimiza(qnt_obj, Volume_obj, Volume_rec):
5
6      for j in range(qnt_obj):  #verifica se os objetos cabem no
       recipiente.
7          if Volume_obj[j]>Volume_rec:
8              print('\n')
9              print(f'Um dos objetos não cabem no recipiente de
       volume {Volume_rec}')
10             return qnt_recipientes
11
12         Aux_volume_rec = Volume_rec
13         qnt_recipientes = 1
14         for i in range(qnt_obj):
15             if Aux_volume_rec >= Volume_obj[i]:
16                 Aux_volume_rec = Aux_volume_rec - Volume_obj[i]
17             else:
18                 qnt_recipientes += 1
19                 Aux_volume_rec = Volume_rec - Volume_obj[i]
20     return qnt_recipientes
```

Para ideia heurística 1:

```
def OTM_shuffle(qnt_entradas, Volume_dos_obj, V):
  k=0
  melhor_sol = 1000000

  while k<10:
    k +=1
    ini = time.time()
    random.shuffle(Volume_dos_obj) #Embaralha o vetor
    sol = func_minimiza(qnt_entradas, Volume_dos_obj, V)
    print(f'{Volume_dos_obj}: {sol}') #Imprime as soluções
   geradas
    if sol < melhor_sol:
      melhor_sol = sol
    fim = time.time()

  print(f'A melhor solução encontrada foi {melhor_sol}')
  print(f'Tempo de execução com k = {k} : {fim-ini} segundos')
```

Para a ideia heurística 2:

```
def OTM_decresce(qnt_entradas, Volume_dos_obj, V):
  ini = time.time()
  Volume_dos_obj.sort(reverse=True)
  sol = func_minimiza(qnt_entradas, Volume_dos_obj, V)
  print(f'{Volume_dos_obj}: {sol}') #Imprime as soluções
   geradas

  fim = time.time()
  print(f'A melhor solução encontrada foi {sol}')
  print(f'Tempo de execução: {fim-ini} segundos')
```

É possível acessar o código em
colab.research.google.com/drive/1Y7HhqMOuUXITidwaGWuNOWZAQkj84gn9?usp=sharing

Testando para esse conjunto

$$[1, 5, 4, 7, 2, 3, 8, 1, 5, 4, 7, 2, 3, 8]$$

A **solução ótima** deste conjunto é 6, porém utilizando as ideias heurísticas obtemos:

```
[7, 1, 5, 3, 3, 2, 2, 8, 7, 4, 1, 5, 4, 8]: 8
[1, 7, 5, 4, 7, 8, 3, 3, 8, 5, 2, 2, 1, 4]: 8
[7, 4, 5, 3, 2, 1, 3, 2, 4, 1, 7, 5, 8, 8]: 8
[1, 7, 3, 5, 4, 7, 8, 2, 2, 8, 5, 3, 1, 4]: 8
[4, 7, 2, 1, 7, 2, 3, 3, 8, 8, 4, 1, 5, 5]: 8
[5, 1, 2, 4, 3, 3, 8, 7, 2, 4, 5, 8, 1, 7]: 7
[5, 8, 3, 4, 5, 2, 1, 3, 7, 1, 4, 7, 8, 2]: 8
[7, 2, 8, 2, 3, 7, 4, 8, 5, 5, 4, 3, 1, 1]: 7
[1, 2, 5, 7, 4, 4, 3, 5, 7, 3, 1, 2, 8, 8]: 8
[7, 2, 1, 7, 8, 3, 5, 4, 3, 1, 5, 2, 8, 4]: 8
A melhor solução encontrada foi 7
```

(a) Ideia Heurística 1

```
[8, 8, 7, 7, 5, 5, 4, 4, 3, 3, 2, 2, 1, 1]: 8
A melhor solução encontrada foi 8
```

(b) Ideia Heurística 2

Note que neste primeiro caso, a *ideia heurística 1* depende de uma probabilidade de ocorrer, porém obtivemos uma solução melhor do que a *ideia heurística 2*. No entanto, agora testaremos para as 10 instâncias. Para melhor análise, tomamos a liberdade de contar o tempo que o algoritmo está rodando, ou seja, analisaremos a complexidade do algoritmo.

Para a primeira instância, temos o seguinte código:

```
with open('Hard28_BPP13.txt', 'r') as f:
    results = [int(line) for line in f.readlines()]


qnt_entradas = results[0]
results.remove(results[0])
results.remove(1000)
Inst_1 = results

OTM_shuffle(qnt_entradas, Inst_1, 1000)
print('\n')
OTM_decresce(qnt_entradas, Inst_1, 1000)
```

Para melhor visualização e manipulação dos dados, o vetor foi suprimido no documento, uma vez que $qnt\_entradas \in [160, 200]$. Sendo assim, para poder mudar as intâncias analisadas, basta mudar o arquivo no código.

Segue abaixo o resultado das *instâncias* utilizando a ideia da **heurística 1** e da **heurística 2**, respectivamente,

```
A melhor solução encontrada foi 91
Tempo de execução com k = 10 : 0.003068685531616211 segundos


A melhor solução encontrada foi 96
Tempo de execução: 0.0039484500885009766 segundos
```

Figura 1: Resultados da Instância 1

```
A melhor solução encontrada foi 78
Tempo de execução com k = 10 : 0.002465963363647461 segundos


A melhor solução encontrada foi 79
Tempo de execução: 0.002435922622680664 segundos
```

Figura 2: Resultados da Instância 2

```
A melhor solução encontrada foi 85
Tempo de execução com k = 10 : 0.002408742904663086 segundos


A melhor solução encontrada foi 88
Tempo de execução: 0.0023391246795654297 segundos
```

Figura 3: Resultados da Instância 3

```
A melhor solução encontrada foi 95
Tempo de execução com k = 10 : 0.0038785934448242188 segundos


A melhor solução encontrada foi 103
Tempo de execução: 0.003374338150024414 segundos
```

Figura 4: Resultados da Instância 4

```
A melhor solução encontrada foi 106
Tempo de execução com k = 10 : 0.004272937774658203 segundos


A melhor solução encontrada foi 113
Tempo de execução: 0.0038461685180664062 segundos
```

Figura 5: Resultados da Instância 5

```
A melhor solução encontrada foi 79
Tempo de execução com k = 10 : 0.0024802684783935547 segundos


A melhor solução encontrada foi 86
Tempo de execução: 0.0022668838500976562 segundos
```

Figura 6: Resultados da Instância 6

```
A melhor solução encontrada foi 91
Tempo de execução com k = 10 : 0.0032494068145751953 segundos


A melhor solução encontrada foi 98
Tempo de execução: 0.004772186279296875 segundos
```

Figura 7: Resultados da Instância 7

```
A melhor solução encontrada foi 99
Tempo de execução com k = 10 : 0.0040340423583984375 segundos


A melhor solução encontrada foi 107
Tempo de execução: 0.003696441650390625 segundos
```

Figura 8: Resultados da Instância 8

```
A melhor solução encontrada foi 111
Tempo de execução com k = 10 : 0.004758596420288086 segundos


A melhor solução encontrada foi 116
Tempo de execução: 0.003506898880004883 segundos
```

Figura 9: Resultados da Instância 9

Já o código da modelagem é dado por:

```
A melhor solução encontrada foi 94
Tempo de execução com k = 10 : 0.003414630889892578 segundos


A melhor solução encontrada foi 100
Tempo de execução: 0.004166841506958008 segundos
```

Figura 10: Resultados da Instância 10

```
def BPP_model(items):

  # Max number of bins allowed.
  qnt_itens = len(items)
  V = 1000

  #Definindo variáveis.
  y = pulp.LpVariable.dicts('BinUsed', range(qnt_itens),
   lowBound = 0, upBound = 1, cat = LpInteger)
  possible_ItemInBin = [(itemTuple[0], binNum) for itemTuple in
    items for binNum in range(qnt_itens)]
  x = pulp.LpVariable.dicts('itemInBin', possible_ItemInBin,
   lowBound = 0, upBound = 1, cat = LpInteger)

  #Definindo o problema como de minimização
  prob = LpProblem("Bin Packing Problem", LpMinimize)

  #Definição da FO
  prob += lpSum([y[i] for i in range(qnt_itens)]), "Objective:
   Minimize Bins Used"

  #Primeira Restrição (O x só pode entrar em 1 Bin)
  for j in items:
    prob += lpSum([x[(j[0], i)] for i in range(qnt_itens)]) ==
   1, ("An item can be in only 1 bin -- " + str(j[0]))

  #Segunda Restrição (A cada x somado, deve dar menos que o
   valor do volume do Bin)
  for i in range(qnt_itens):
    prob += lpSum([items[j][1] * x[(items[j][0], i)] for j in
   range(qnt_itens)]) <= V*y[i], ("The sum of item sizes must be
    smaller than the bin -- " + str(i))

  #Começando a contar no relógio o solve
  start_time = time.time()
```

```
28
29    prob.solve()
30    print("Bins used: " + str(sum(([y[i].value() for i in range(
       qnt_itens)]))))
31    print("Solved in %s seconds." % (time.time() - start_time))
```

```
1  with open('Hard28_BPP13.txt', 'r') as f:
2      results = [int(line) for line in f.readlines()]
3
4  qnt_entradas = results[0]
5  results.remove(results[0])
6  results.remove(1000)
7  Inst_1 = results
8
9  lista2=[]
10 for i in range(len(Inst_1)):
11   x = tuple([i, Inst_1[i]])
12   lista2.append(x)
13
14 print(lista2)
15
16 BPP_model(lista2)
```

(Basilyan; Michael. 2015)

Todas as instâncias podem ser vistas em: https://drive.google.com/drive/folders/1dySClS3-MNq50yGv8tpdWIPJFmHgzT1k?usp=sharing. Assim como os códigos: https://colab.research.google.com/drive/1GwwD34-2UFoVfBsWlH-naAXf1pHOtigl?usp=sharing. Além disso, utilize https://codepy.io/user/public/DraftBeginExact/. para limitar o tempo.

Note que se compararmos as duas ideias heurísticas, destes o melhor resultado foi dado pela **heurística 1** com apenas 10 mudanças na lista. Como as soluções são probabilísticas, se aumentarmos o número de vezes que embaralhamos o vetor, maior será a chance de encontrarmos uma solução ótima, porém o tempo de processamento dos dados também aumentam significativamente. Se observarmos o tempo de execução do código em ambos os casos estão bem próximos.

Para facilitar a comparação entre os resultados obtidos, temos abaixo uma tabela com as soluções encontradas:

Instância	1	2	3	4	5	6	7	8	9	10
Hard28_BBP	13	40	60	144	178	14	47	119	175	181
Heurística 1	91	78	85	95	106	79	91	99	111	94
Heurística 2	96	79	88	103	113	86	98	107	116	100
Modelo	71	63	68	80	87	66	77	82	89	78
Solução Ótima	67	59	63	73	81	62	71	77	84	72

De todos, o maior tempo de processamento foi dado pelo **código do modelo**, assim como a **melhor solução.** Como vimos anteriormente o problema do empacotamento é tido como NP, ou seja, sua solução para uma entrada significativa, o tempo de excução é bem longo, por esse motivo tivemos que limitar o seu tempo em $260s$. Além disso, a elaboração do código da modelagem é mais complexo e difícil que o das ideias heurísticas.

Portanto, concluímos que o erro das soluções heurísticas são maiores que a do modelo, no entanto, são bem fáceis de serem modeladas.

REFERÊNCIAS

- Delorme,C; Iori, M.; Martello, S. BPPLIB: Uma biblioteca para problemas de embalagem e corte de estoque. Cartas de otimização, 2018. Disponível em: :http://or.dei.unibo.it/library/bpplib. Acesso em: 5 de dez. 2020.
- MIYAZAWA, Flávio Keidi. Problemas de Corte e Empacotamento. 2020. Disponível em: https://www.ic.unicamp.br/~fkm/problems/empacotamento.html. Acesso em: 10 dez. 2020.
- Basylian; Michael. Bin Packing in Python with PuLP. Dez 23, 2015. Disponível: https://www.linkedin.com/pulse/bin-packing-python-pulp-michael-basilyan Acesso: 17 dez. 2020