Análise de Algoritmos Bin Packing com Heurística First Fit

Lucas Borges Fernandes
Gustavo Silveira Marques

Bin Packing

O problema do empacotamento consiste em armazenar um conjunto finito de pacotes em um número finito de caixas utilizando o menor número possível de caixas. Este problema está no conjunto dos problemas NP-hard.

Cada pacote P possui um volume V e cada caixa possui uma capacidade de armazenamento CA.

Um pacote P cabe numa caixa C se a capacidade restante da caixa é maior ou igual a V.

Não há pacote com volume maior que a capacidade de armazenamento da caixa.

Bin Packing (Abordagem exponencial)

A maneira de se encontrar a resposta ótima para o problema é testar todas as possibilidades de ordem de empacotamento para uma quantidade Q de caixas que varia de 1 até N, em que N é a quantidade de pacotes. A primeira quantidade de caixas que o empacotamento se completar é a resposta desejada.

```
1  Seja E o conjunto com todas as permutações do vetor de volume de pacotes VP.
2  Seja CA a capacidade de armazenamento da caixa.
3  Seja N a quantidade de pacotes.
4
5  para Q de 1 até N faça:
6      para cada C em E faça:
7          Seja BIN um vetor de inteiros com Q posições inicialmente zeradas.
8          Seja qtdCx um inteiro inicialmente com valor igual a 1.
9          Seja fail um booleano inicialmente com valor igual false.
10         para cada p em C faça:
11             Seja packed um booleano inicialmente com valor igual a false
12             para cx de 1 até qtdCx faça
13                 se BIN[ cx ] + p <= CA então:
14                     BIN[ cx ] <- BIN[ cx ] + p
15                     packed <- true
16                     break;
17                 fim se
18             fim para
19             se !packed então
20                 se qtdCx + 1 <= Q
21                     qtdCx <- qtdCx + 1
22                     BIN[ qtdCx ] <- p
23                 senão
24                     fail <- true
25                     break;
26                 fim se
27             fim se
28         fim para
29         se !fail então
30             retorne Q
31         fim se
32     fim para
33 fim para
```

Análise do algoritmo:

Linhas 1-3: custo constante

Linha 5: custo O(N)

Linha 6: custo O(N!)

Linhas 7-9: custo constante

Linha 10: custo O(N)

Linha 11: custo constante

Linha 12: custo O(qtdCx)

Linhas 13-33: custo constante

Custo total do algoritmo:

$O(N * N * M * N!) = O(N^2 * M * N!)$,

em que é M a quantidade de caixas utilizadas e N a quantidade de pacotes.

Bin Packing (First fit)

Uma das maneiras de resolver o problema do empacotamento é utilizar a heurística conhecida como *first fit*.

Nesta heurística para cada pacote P a ser empacotado, percorre-se todas as caixas já utilizadas e coloca-o na primeira caixa que o couber. Se não houver caixa que o caiba, abre-se uma nova caixa e o insere nela.

```
1  Seja VP um vetor de inteiros positivos que representam os volumes dos pacotes.
2  Seja CA a capacidade de armazenamento da caixa.
3  Seja N a quantidade de pacotes.
4
5  para cada v em VP faça:
6      Seja packed um booleano inicialmente com valor igual a false
7      para cx de 1 até qtdCx faça
8          se BIN[ cx ] + v <= CA então:
9              BIN[ cx ] <- BIN[ cx ] + p
10             packed <- true
11             break;
12         fim se
13     fim para
14     se !packed então
15         qtdCx <- qtdCx + 1
16         BIN[ qtdCx ] <- p
17     fim se
18 fim para
19 retorne qtdCx
```

Análise do algoritmo:

Linhas 1-3: custo constante

Linha 5: custo O(N)

Linha 6: custo constante

Linha 7: custo O(qtdCx)

Linhas 8-19: custo constante

Custo total do algoritmo:

O(N * M), em que é M a quantidade de caixas utilizadas e N a quantidade de pacotes.

First fit: razão de aproximação

No cenário ideal de empacotamento cada caixa será utilizada no máximo de sua capacidade. Neste caso há um limite inferior, Linf, para a quantidade de caixas a serem utilizadas, que é obtido através da divisão:

$$Linf = \left\lceil \left(\sum_{i=1}^{P} P_i\right) / CA \right\rceil$$

Logo temos que a solução ótima OPT >= Linf.

First fit: razão de aproximação

Pela maneira com que o algoritmo constrói a solução, podemos afirmar que existe no máximo uma caixa (e neste caso seria a última caixa) cuja ocupação seja menor ou igual a CA / 2. Em outras palavras, **se forem utilizadas N caixas existem, pelo menos, N - 1 caixas com ocupação maior que CA / 2.**

Isto se deve ao fato de que se houvesse duas caixas com volume menor ou igual a CA / 2, estas poderiam ser agrupadas em uma só, situação que não ocorre já que o algoritmo procura sempre alguma caixa que caiba o item.

First fit: razão de aproximação

Então podemos afirmar que: uso(C_i) > CA / 2, para todo i variando de 1 a N - 1.

De forma que se somarmos os valores ocupados de cada uma das N - 1 caixas teremos um valor maior que (N - 1) * CA / 2, ou seja:

$$\sum_{i=1}^{N-1} C_i > \sum_{i=1}^{N-1} CA/2 \Rightarrow \sum_{i=1}^{N-1} C_i > (N-1)^* CA/2$$

Mas se somando o uso de apenas N - 1 caixas a inequação já é válida, se somarmos todas as caixas, com mais razão ainda a inequação é verdadeira.

First fit: razão de aproximação

Então podemos afirmar que: $\sum_{i=1}^{N} C_i > (N-1) * CA/2$

Podemos ainda fazer mais uma modificação: somar o uso de todas as caixas dá no mesmo que somar todos os volumes dos pacotes.

$$\sum_{i=1}^{P} P_i > (N-1) * CA/2$$

Sendo P a quantidade de pacotes e P_i o peso do pacote i.

First fit: razão de aproximação

Se multiplicarmos os dois lados por 2 / CA, vamos ter que:

$$2 * (\sum_{i=1}^{P} P_i) / CA > (N-1)$$

Nota-se que o lado esquerdo da inequação se assemelha ao Linf obtido anteriormente. Como o Linf é o teto da divisão do somatório por CA, temos que 2 * Linf é maior que o lado esquerdo da inequação, o que nos permite dizer que:

$$2 * Linf > (N-1)$$

First fit: razão de aproximação

Como OPT >= Linf, podemos dizer que: $2 * OPT > (N-1)$

E se 2 * OPT > (N - 1) então 2 * OPT >= N, logo:

$$2 \geq N/OPT$$

E então concluímos que a razão de aproximação da heurística *First Fit* é, pelo menos, 2.

Comparação de execução