

1D - Bin Packing

Next Fit

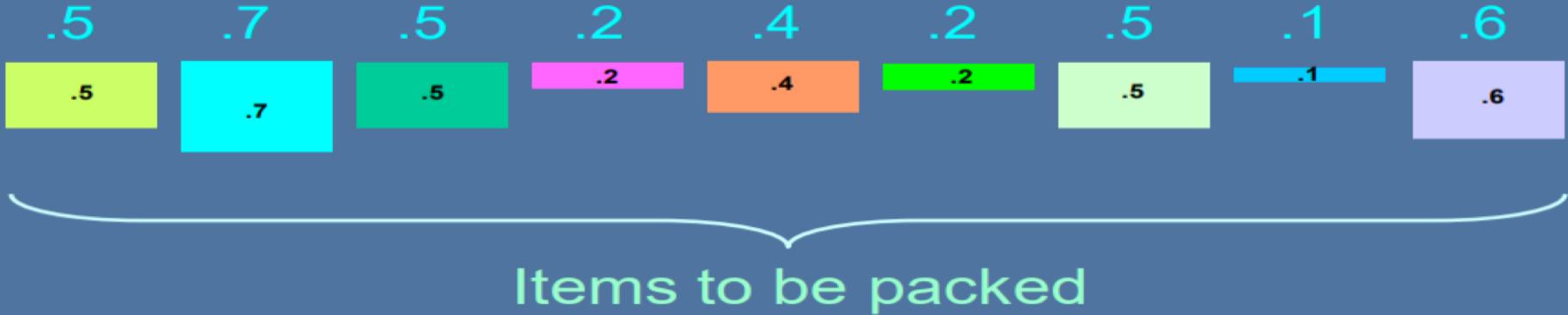
Adriana Peres
João Lucas Flauzino
Jocival Dias

Bin Packing Problem

Dado um conjunto $I = \{1, \dots, n\}$ de itens, onde $i \in I$ e tem tamanho $s_i \in (0, 1]$ e um conjunto $B = \{1, \dots, n\}$ de bins (caixas) com capacidade 1.

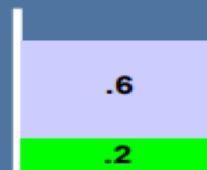
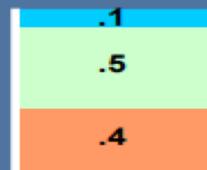
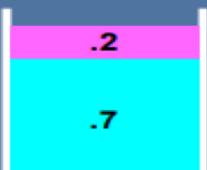
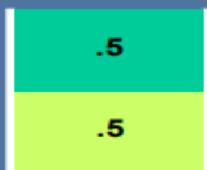
Encontrar uma maneira de agrupar os elementos em bins de forma que a quantidade de bins utilizados seja mínima.

Bin Packing (1-D)



Bin Packing Problem

.5 .7 .5 .2 .4 .2 .5 .1 .6



$$N_0 = 4$$

Bin Packing - Aplicações

O problema do empacotamento é comum em várias áreas do conhecimento (e.g. Ciência da Computação, Logística e Planejamento). Algumas de suas aplicações são:

1. Alocar dados de backup em múltiplas fitas;
2. Job scheduling;
3. Colocar propagandas em intervalos fixos na TV;
4. Carregar Caminhões com caixas (containers);

Bin Packing - Online e Offline

Online: Classe de problemas onde cada elemento será atribuído a uma caixa (e não será alterado novamente) e o próximo item só é conhecido após a alocação do anterior.

Offline: Classe de problemas onde todos os itens são conhecidos e podem ser analisados antes da atribuição em suas respectivas caixas.

A maioria das aplicações reais do Bin Packing tratam de algoritmos offline (e.g. recorte).

Bin Packing Problem - NP-completo

O problema do empacotamento é NP-completo. Não existe algoritmo em tempo polinomial para encontrar sua solução.

Mais especificamente, é NP-completo decidir se uma instância do Bin Packing admite uma solução com apenas dois bins.

Prova -> Se dá pela redução do problema de Partição, que é NP-Completo. Dada uma instância do problema de Partição, uma instância para o BIN PACKING é criada atribuindo $s_i = 2c_i / \sum_{j=1}^n c_j \in (0, 1]$ para $i = (0, \dots, n)$. Dois bins serão suficiente se e somente se existe um $S \subseteq \{1, \dots, n\}$ que $\sum_{i \in S} c_i = \sum_{i \notin S} c_i$

Bin Packing - Monotonic

Um algoritmo para a resolução do problema Bin Packing é **Monotonic** se o número de bins que é utilizado para uma lista X é sempre maior do que uma sublista de X.

Next Fit - É *monotonic* pois, se ele não fosse, então teria uma instância Z tal que removendo um item levaria a adição de bins. Porém, é possível perceber que essa situação é impossível. Seja i , o item que foi removido, os itens $i' < i$ continuaram em seus respectivos bins e os itens $i'' > i$ podem apenas ser movidos para bins anteriores (ou mantidos no mesmo).

Bin Packing - α -aproximação

Teorema: A menos que $P = NP$, não existe α -aproximação para o bin packing com $\alpha < 3 / 2$.

Existem algoritmos (PD) que resolvem o Bin Packing com fator de aproximação menor que $3 / 2$ porém com número limitado de tamanhos de itens distintos e um número limitado de itens cabem em um bin.

Bin Packing - Brute Force

Dado um conjunto $I = \{1, \dots, n\}$ de itens, onde $i \in I$ e tem tamanho $s_i \in (0, 1]$ e um conjunto $B = \{1, \dots, n\}$ de bins (caixas) com capacidade 1.

Gere todas as permutações possíveis do arranjo I e avalie, de forma a minimizar, a quantidade de bins utilizado. A solução ótima é obtida através da permutação que utilizar a menor quantidade de bins.

O número ótimo de soluções (OPT) é limitado inferiormente por:

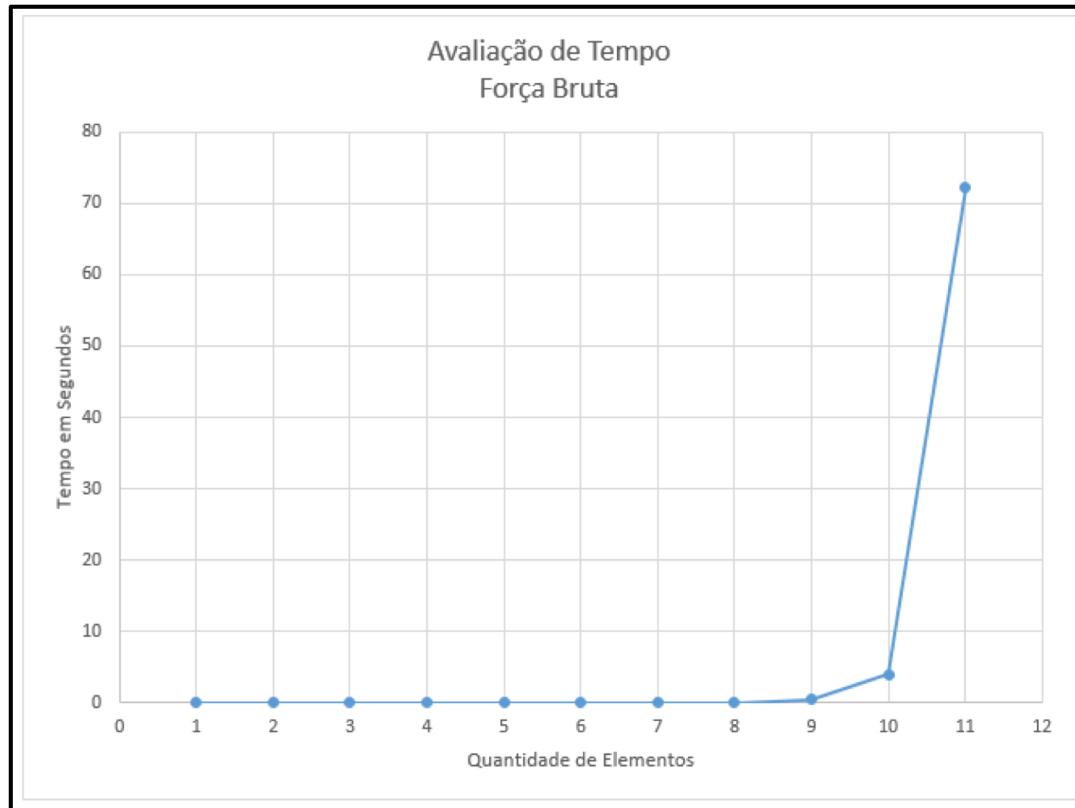
$$\left\lceil \sum_{i=1}^n s_i / \text{capacidade(bins)} \right\rceil$$

Bin Packing - Brute Force

Tempo do algoritmo: O(n.n!)

```
algoritmo brute_force(I, n)
01. R <- []
02. foreach permutação(I) do
03.     R' <- []
04.     for i = 1 to n
05.         if cabe(bin_atual, I[i])
06.             then empacota(bin_atual, I[i])
07.         else
08.             fecha(bin_atual)
09.             insere(R, bin_atual)
10.             bin_atual <- cria_novo_bin()
11.             empacota(bin_atual, I[i])
12.         if nro_bins(R') < nro_bins(R)
13.             then R <- R'
14. return R
```

Bin Packing - Brute Force



Bin Packing - Next Fit

Dado um conjunto $I = \{1, \dots, n\}$ de itens, onde $i \in I$ e tem tamanho $s_i \in (0, 1]$ e um conjunto $B = \{1, \dots, n\}$ de bins (caixas) com capacidade 1.

Avalie para cada item se o mesmo cabe no bin atual. Caso seja possível empacotar o elemento neste bin, caso contrário, feche o bin anterior, abra um novo bin e empacote o elemento no novo bin.

O número ótimo de soluções (**OPT**) é limitado inferiormente por:

$$\left\lceil \sum_{i=1}^n s_i / \text{capacidade}(bins) \right\rceil$$

Bin Packing - Next Fit

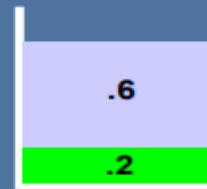
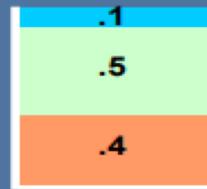
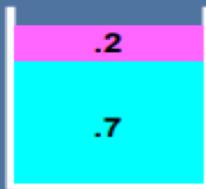
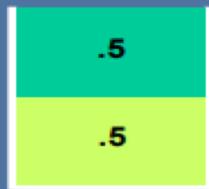
Tempo do algoritmo: $O(n)$

Algorithm Next-Fit

```
1: for All objects  $i = 1, 2, \dots, n$  do
2:   if Object  $i$  fits in current bin then
3:     Pack object  $i$  in current bin.
4:   else
5:     Create new bin, make it the current bin, and pack object  $i$ .
6:   end if
7: end for
```

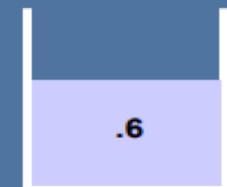
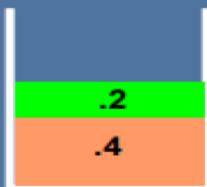
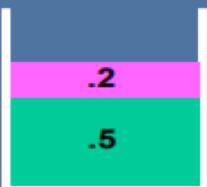
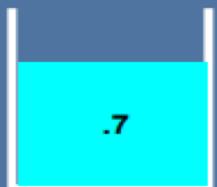
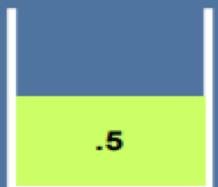
Bin Packing Problem

.5 .7 .5 .2 .4 .2 .5 .1 .6



$$N_0 = 4$$

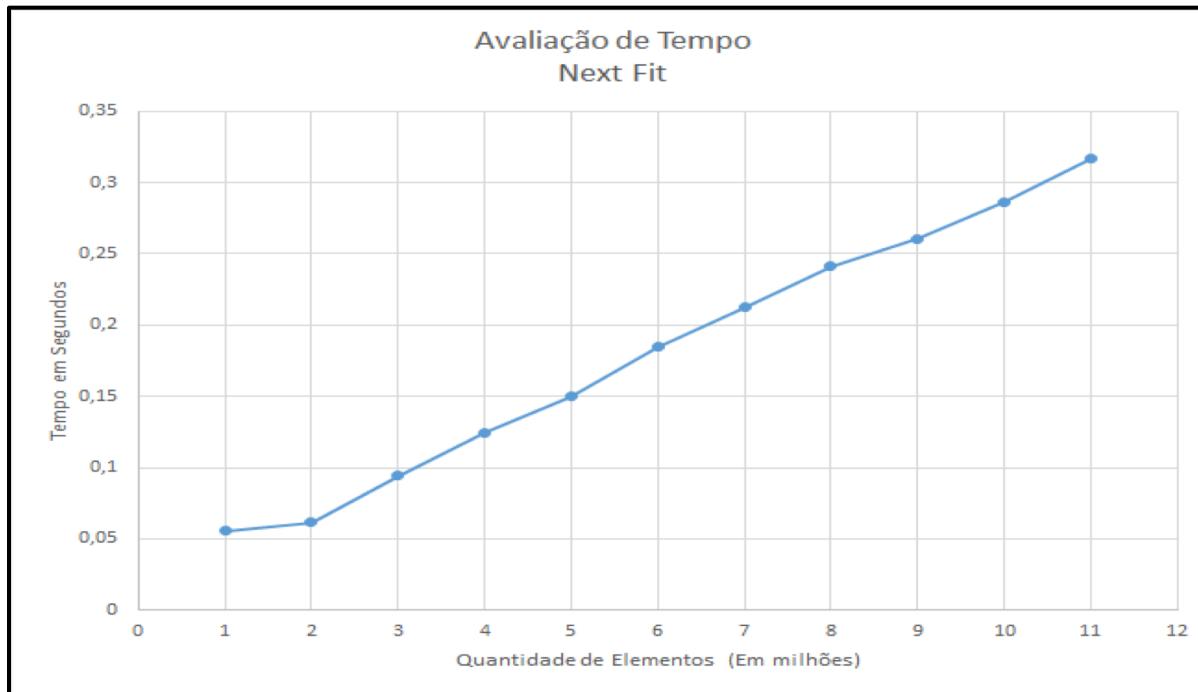
Next Fit Packing Algorithm



$$N = 6$$

$$\frac{N}{N_0} \leq 2$$

Next Fit - Algoritmo Aproximado



Bin Packing - Next Fit - α -aproximação

O algoritmo Next-Fit não utiliza mais do que 2 vezes o valor OPT.

Isso se dá ao fato que para dois bins adjacentes a soma dos seus pesos combinados não pode ser menor do que 1 pois, caso fosse, os conteúdos do segundo bin seriam colocados no primeiro.

Consequentemente a soma combinada de todos os bins não será menor do que metade da capacidade total dos bins, então não será necessário mais do que $2 * \text{OPT}$ bins.

Bin Packing - Next Fit - α -aproximação - prova

Para N par temos:

$$s(B1) + s(B2) \geq 1$$

$$s(B3) + s(B4) \geq 1$$

...

$$s(Bn-1) + s(Bn) \geq 1$$

$$\sum s(bins) \geq n/2$$

$$n \leq 2 * \sum s(bins) \leq 2 * \left\lceil \sum s(bins) \right\rceil \leq 2 * OPT$$

Bin Packing - Next Fit - α -aproximação - prova

Para N ímpar temos:

$$s(B1) + s(B2) \geq 1$$

$$s(B3) + s(B4) \geq 1$$

...

$$s(Bn-2) + s(Bn-1) \geq 1$$

$$\sum s(bins) \geq (n-1)/2 + s(Bn)$$

$$2 * \sum s(bins) \geq [n-1 + 2 * s(Bn)]$$

$$2 * \sum s(bins) \geq n$$

$$n \leq \left\lceil 2 * \sum s(bins) \right\rceil \leq 2 * \left\lceil \sum s(bins) \right\rceil \leq 2 * OPT$$

Bin Packing - Next Fit - α -aproximação

Portanto, temos que o Next Fit é aproximado em 2 vezes o valor da solução ótima. Essa aproximação é obtida devido ao fato do algoritmo garantir que nenhum bin fique com menos que metade da sua capacidade possível.

Pior caso

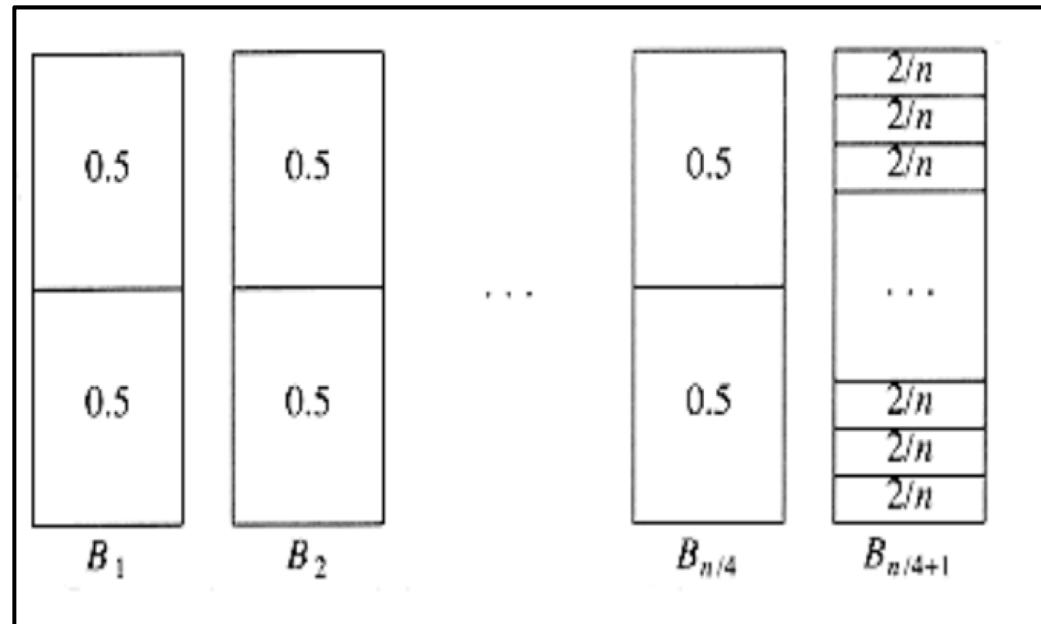
O pior caso seria composto por $4n$ itens dispostos da seguinte maneira:
 $(\frac{1}{2}, \frac{1}{2n}, \frac{1}{2}, \frac{1}{2n}, \dots, \frac{1}{2}, \frac{1}{2n})$.

A solução ótima, $\text{OPT} = n+1$, enquanto Next Fit retorna $2*n$.

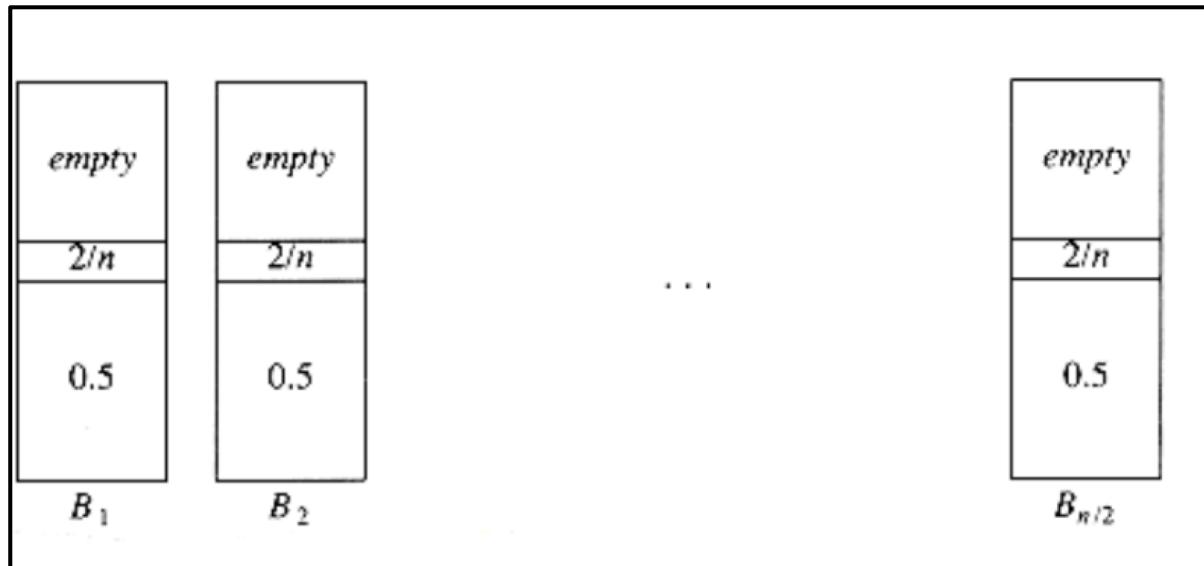
A solução ótima irá colocar 2 itens com peso $\frac{1}{2}$ juntos em n bin e $2*n$ itens com peso $\frac{1}{2} * 1/n$ juntos em um bin, usando $n+1$ bins.

Next Fit irá colocar 2 itens adjacentes juntos, necessitando de $2*n$ bins.

Pior caso - Solução ótima



Pior caso - Next Fit



Implementação - Força Bruta

```
tic;
qtd_itens = 11;
wmax = 1;
itens = rand(qtd_itens,1);
minimo = Inf;
limite_minimo = ceil(sum(itens(:)) / wmax);
maximo = qtd_itens;
peso_pacote_minimo = zeros(1,maximo);
permutacao = perms(itens);
iteracoes = size(permutacao,1);

for z = 1:iteracoes
    nro_pacotes = 1;
    perm_itens = permutacao(z, :);
    peso_pacotes = zeros(1,qtd_itens);
    for ii = 1:qtd_itens
        if(perm_itens(ii) < (wmax-peso_pacotes(nro_pacotes)))
            peso_pacotes(nro_pacotes) = peso_pacotes(nro_pacotes) + perm_itens(ii);
        else
            nro_pacotes = nro_pacotes + 1;
            peso_pacotes(nro_pacotes) = peso_pacotes(nro_pacotes) + perm_itens(ii);
        end
    end
    if nro_pacotes < minimo
        minimo = nro_pacotes;
        peso_pacote_minimo = peso_pacotes(1:nro_pacotes);
    end
end

toc;
minimo
peso_pacote_minimo
```

Implementação - Algoritmo aproximado

```
close all;
tic;
qtd_itens = 11000000;
wmax = 1;
itens = rand(qtd_itens,1);
limite_minimo = ceil(sum(itens(:)) / wmax);

nro_pacotes = 1;
peso_pacotes = zeros(1,qtd_itens);
for ii = 1:qtd_itens
    if(itens(ii) < (wmax-peso_pacotes(nro_pacotes)))
        peso_pacotes(nro_pacotes) = peso_pacotes(nro_pacotes) + itens(ii);
    else
        nro_pacotes = nro_pacotes + 1;
        peso_pacotes(nro_pacotes) = peso_pacotes(nro_pacotes) + itens(ii);
    end
end

toc;
nro_pacotes
double(nro_pacotes)/limite_minimo
```

Referências

1. http://ac.informatik.uni-freiburg.de/lak_teaching/ws11_12/combopt/notes/bin_packing.pdf
2. <http://www.or.deis.unibo.it/kp/Chapter8.pdf>
3. <http://orion.lcg.ufrj.br/Dr.Dobbs/books/book3/chap10.htm>
4. <http://community.wvu.edu/~krsubramani/courses/sp01/approx/lecnotes/lecture13.pdf>
5. <https://www.it.uu.se/edu/course/homepage/realtid/ht11/schedule/bin-packing.pdf>
6. <http://www.cs.ucsb.edu/~teo/cs130b.w15/bp.pdf>
7. <http://www.or.deis.unibo.it/kp/Chapter8.pdf>

Todos retirados em 08/12/2018 às 17:00.