Algoritmos de Empacotamento Tridimensional

novas estratégias e análises de desempenho

Flávio Keidi Miyazawa

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Área de Concentração: **Ciência da Computação**
Orientador: **Profa. Dra. Yoshiko Wakabayashi**

Durante a elaboração deste trabalho o autor recebeu apoio financeiro da CAPES

-São Paulo, novembro de 1993-

Algoritmos de Empacotamento Tridimensional

novas estratégias e análises de desempenho

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Flávio Keidi Miyazawa
e aprovada pela comissão julgadora.

São Paulo, 26 de janeiro de 1994.

Banca examinadora:

- Prof. Dra. Yoshiko Wakabayashi (orientadora) - IME-USP
- Prof. Dr. Arnaldo Mandel - IME-USP
- Prof. Dr. João Meidanis - IMECC-UNICAMP

à minha irmã Miriam
e aos meus pais Marina e Takuo

Resumo

Nesta dissertação estudamos dois tipos de problemas de empacotamento tridimensional. Um dos problemas constitui a versão tradicionalmente chamada de *empacotamento tridimensional ortogonal*. O outro problema refere-se à versão que denominamos de *empacotamento tridimensional ortogonal e orientado na dimensão z*. A diferença entre esta versão e a anterior reside no fato de que nesta é permitido fazer um certo tipo de rotação das caixas.

Além dos dois problemas gerais, estudamos também vários casos particulares desses problemas, obtidos de acordo com as restrições sobre as formas e os tamanhos das caixas a serem empacotadas.

Apresentamos vários algoritmos de aproximação para estes problemas e analisamos o desempenho assintótico desses algoritmos. Incluimos aqui resultados encontrados na literatura e outros que obtivemos, melhorando alguns resultados referentes à qualidade dos limites de desempenho assintótico dos algoritmos.

Abstract

In this dissertation we study two versions of three-dimensional packing problems. One of the versions is the so-called *orthogonal three-dimensional packing problem*. The other version is referred here as the *orthogonal z-oriented three-dimensional packing problem*. The difference between these versions is that the last one allows the boxes to be rotated, while in the first case this is not allowed.

Besides these two general problems, we also study some special cases of these problems, which arise when restrictions are imposed on the shape and the size of the boxes to be packed.

We present many approximation algorithms for these problems and analyse them with respect to their asymptotic performance. Results known in the literature as well as new ones are presented. We describe a number of new algorithms whose asymptotic performance ratio compares favourably to those known in the literature.

Agradecimentos

À Yoshiko, pela super orientação, dedicação, paciência, e todo esforço para que esta dissertação assumisse esta forma final.

À Aurea, por todos os momentos que passamos juntos.

Ao Haroldo, Orlando, Ronaldo, Álvaro e Maria Angela pelos seminários, discussões e interesse.

A todos do Departamento de Ciência da Computação por "terem me quebrado aquele galho".

Ao Ricardo Ueda pelos empurrões iniciais.

Ao Arnaldo e ao Siang pelos conselhos no início do curso.

À Cristina Fernandes por me emprestar o material do exame de qualificação.

Ao Carlinhos e ao Mário pelos artigos conseguidos.

Às pessoas que fizeram este mestrado mais divertido: Cezar, Paulo Pinheiro, Edson M., Alfredo, Humberto, Fátima, Andrea Z., Fábio M., Fábio K., Marcelo, Glauber, Bira e Leojb.

Finalmente, aos meus pais que sempre me incentivaram a estudar.

Introdução

Problemas de empacotamento, nas suas diversas versões, têm sido amplamente estudados, principalmente devido à sua aplicabilidade no tratamento de problemas que ocorrem tanto na computação quanto na indústria de manufatura.

A denominação genérica de "problema de empacotamento" se aplica aos problemas que, na sua forma geral, requerem que certos objetos sejam 'empacotados' (ou 'cortados') em outros de 'mesmo tipo e de tamanho maior'. Dependendo do tipo de objeto (barra, placas, caixas), temos os chamados problemas de empacotamento unidimensional, bidimensional e tridimensional.

Mesmo para cada uma dessas versões, existem diversas variantes, de acordo com o que se procura otimizar. As versões do caso unidimensional, apesar de aparentemente mais simples, são computacionalmente difíceis e têm sido investigadas desde o início da década de 70. Vários algoritmos de aproximação têm sido desenvolvidos para este caso e também para o caso bidimensional. Já o estudo de problemas de empacotamento tridimensional é mais recente.

Diferentes tipos de análises têm sido feitas quanto ao desempenho dos algoritmos de empacotamento. A maioria destas análises se baseia na comparação do valor obtido pelo algoritmo com o valor do empacotamento ótimo. As principais linhas de análise seguidas são: combinatoriais (ou análises de pior caso), probabilísticas e experimentais.

As análises *combinatoriais* se baseiam na garantia de um desvio máximo (em relação à solução ótima) da solução aproximada obtida por uma determinada heurística quando aplicada para uma dada classe de instâncias.

Nas análises *probabilísticas*, assume-se uma função de densidade para as instâncias do problema e estabelece-se propriedades probabilísticas da heurística. Estuda-se o desempenho esperado da heurística ou um limite para a probabilidade de a heurística encontrar uma solução dentro de uma percentagem de optimalidade pré-determinada.

Por fim, as análises *experimentais* comparam experimentalmente o desempenho de vários algoritmos quando aplicados a determinadas instâncias do problema.

Nesta dissertação estudamos duas versões de problemas de empacotamento tridimensi-

onal. A versão tradicional foi chamada aqui simplesmente de *Problema do Empacotamento Tridimensional (PET)*. Trata-se mais especificamente do caso ortogonal e orientado nas três dimensões. A outra versão foi chamada aqui de *Problema do Empacotamento Tridimensional e Orientado na Dimensão z* (PET^R). A diferença entre esta versão e a anterior reside no fato de que nesta é permitido fazer um certo tipo de rotação das caixas.

Apresentamos vários algoritmos de aproximação para estes dois problemas e fazemos análises combinatoriais de desempenho.

No **primeiro capítulo** definimos o *Problema do Empacotamento Tridimensionsal* e estabelecemos a notação e as várias medidas de desempenho que são usadas na análise dos algoritmos. Apresentamos também algumas definições básicas.

No **segundo capítulo** estudamos problemas de empacotamento unidimensional e bidimensional. Aqui mostramos como as principais versões para estes dois problemas podem ser vistas como casos especiais do Problema do Empacotamento Tridimensional.

O objetivo deste capítulo não é descrever todos os algoritmos existentes para estes problemas. Apresentamos aqueles que são utilizados, no capítulo seguinte, como subrotina de algoritmos para o Problema do Empacotamento Tridimensional. Devido à complexidade e extensão das demonstrações, apenas enunciamos a maioria dos resultados que são usados na análise dos algoritmos. Em alguns casos apresentamos a prova dos resultados mencionados.

No **terceiro capítulo** estudamos o Problema do Empacotamento Tridimensional. Os principais resultados para este problema são bem recentes e foram obtidos por Li e Cheng [22, 23, 25]. Estes autores consideraram também casos especiais deste problema quando há restrições sobre o fundo das caixas a serem empacotadas. Apresentamos alguns algoritmos novos que desenvolvemos para o problema e para estes casos especiais. Analisamos o desempenho dos algoritmos propostos e mostramos que alguns deles têm limites de desempenho assintótico melhores que os dos algoritmos de Li e Cheng.

No **quarto capítulo** investigamos o *Problema do Empacotamento Tridimensional Ortogonal Orientado na Dimensão z*. Este problema é tão complicado quanto o problema que não permite rotacionar as caixas. Apresentamos alguns algoritmos para esta variante que foram desenvolvidos fazendo-se modificações nos algoritmos vistos no Capítulo 3.

Finalmente, apresentamos quatro apêndices.

No **Apêndice A** estudamos a complexidade computacional do Problema do Empacotamento Tridimensional e de alguns de seus casos particulares. Mostramos que tanto o caso geral como vários casos particulares são $\mathcal{NP}$-difíceis. Esses resultados foram encontrados na literatura.

No **Apêndice B** apresentamos um resumo sobre um problema muito parecido com o Problema do Empacotamento Tridimensional, o *Problema do Empacotamento Tridi-*

mensional em Caixas. Aqui apenas citamos os resultados principais e indicamos algumas referências.

No **Apêndice C** listamos algumas aplicações dos problemas de empacotamento. Além das aplicações no caso tridimensional, com ou sem rotação sobre o fundo, listamos também aplicações nos casos uni- e bidimensional.

Por fim, no **Apêndice D** apresentamos uma tabela contendo os algoritmos de empacotamento tridimensional estudados nesta dissertação, e os seus respectivos limites de desempenho assintótico.

Conteúdo

Capítulo 1

Notações e Preliminares

Neste capítulo apresentaremos os conceitos básicos que serão usados ao longo desta dissertação.

Na primeira seção definiremos o problema a ser estudado e estabeleceremos a notação. Na segunda seção apresentaremos algumas medidas de desempenho de algoritmos de aproximação, que serão utilizadas no presente trabalho.

1.1 Conceitos básicos

Nosso estudo trata do problema de dispor caixas retangulares de diferentes tamanhos dentro de uma caixa também retangular de tamanho fixo, de tal forma que não haja sobreposição das primeiras e a altura total desta disposição seja mínima.

A definição informal acima, além de ser vaga, não serve para um tratamento mais rigoroso do problema. Passaremos então a definir formalmente os objetos de nosso estudo.

Fixaremos um sistema de coordenadas tridimensional (x, y, z) com origem $(0, 0, 0)$. Uma caixa será representada por uma tripla $c = (w, l, h)$ onde w, l e h são a **largura**, o **comprimento** e a **altura** da caixa c, respectivamente. Uma notação comumente usada para indicar a largura, o comprimento e a altura de uma caixa c será $\mathbf{x}(\mathbf{c}), \mathbf{y}(\mathbf{c})$ e $\mathbf{z}(\mathbf{c})$, respectivamente. Convencionaremos que se c_i é uma caixa, mesmo que não haja menção explicita, $c_i = (x_i, y_i, z_i)$.

Nos problemas de nosso interesse suporemos que a caixa *grande*, na qual serão empacotadas todas as outras caixas, sempre tem altura suficiente para englobar qualquer quantidade de caixas. Uma tal caixa será denotada por $B = (a, b, \infty)$, onde $a, b \in \mathbb{R}_+^*$. Neste caso, B é uma caixa de largura a, comprimento b e altura *ilimitada* (veja a figura

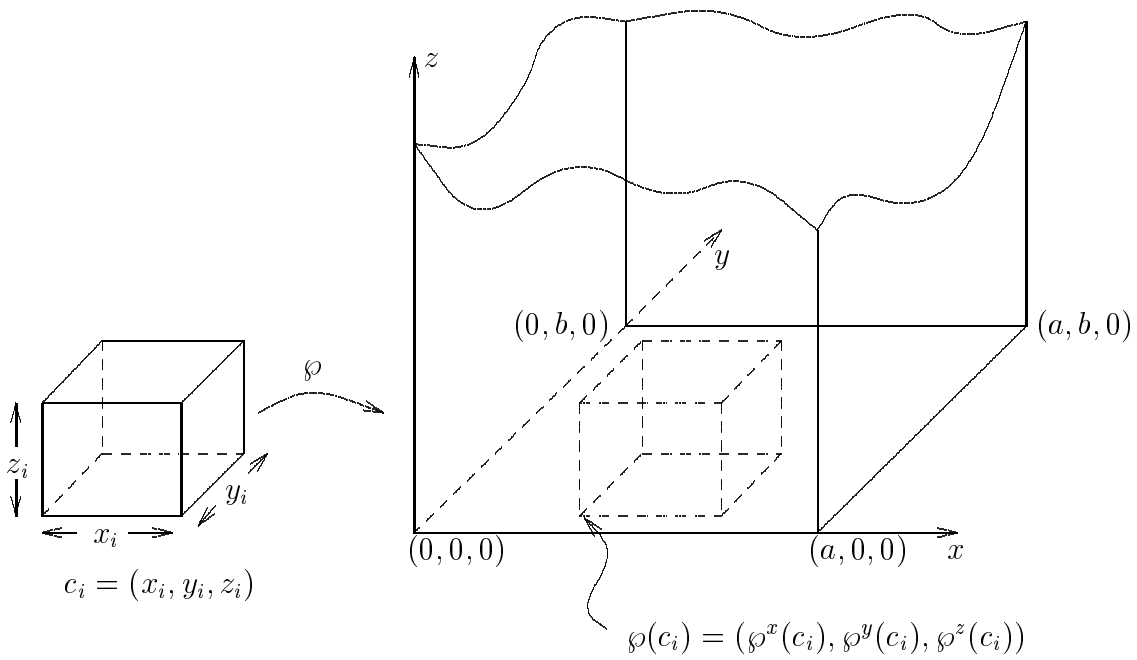


Figura 1.1: Empacotamento de uma caixa $c_i = (x_i, y_i, z_i)$ em uma caixa $B = (a, b, \infty)$.

1.1). Convencionaremos que B corresponde à região

$$[0, a) \times [0, b) \times [0, \infty).$$

Considere uma caixa $B = (a, b, \infty)$ e uma lista de caixas $L = (c_1, c_2, \ldots, c_n)$. Lembramos que $c_i = (x_i, y_i, z_i),\ i = 1, \ldots, n$, como mencionado. Um **empacotamento** $\wp$ de L em B é um mapeamento $\wp : L \to B$ tal que

$$\wp^x(c_i) + x_i \leq a \quad \text{e} \quad \wp^y(c_i) + y_i \leq b\ ,$$

onde $\wp(c_i) = (\wp^x(c_i), \wp^y(c_i), \wp^z(c_i)),\ i = 1, \ldots, n$.

Além do mais, se $\mathbf{R(c_i)}$ é definido como

$$R(c_i) = [\wp^x(c_i), \wp^x(c_i) + x_i) \times [\wp^y(c_i), \wp^y(c_i) + y_i) \times [\wp^z(c_i), \wp^z(c_i) + z_i),$$

então

$$R(c_i) \cap R(c_j) = \emptyset, \quad \forall i, j,\ 1 \leq i \neq j \leq n\ ,$$

i.e., duas caixas em L não podem se sobrepor em um empacotamento $\wp$.

Note que o empacotamento definido é ortogonal e orientado nas três dimensões. Ou seja, cada caixa em L deve ser guardada em B ortogonalmente em relação às três dimensões e de modo que os eixos que definem a largura, o comprimento e altura de c fiquem paralelos aos eixos x, y e z, respectivamente. Mais informalmente, isto significa que as caixas de L não podem ser rotacionadas, nem 'tombadas de lado', nem viradas 'de cabeça para baixo'.

Define-se a **altura** de um empacotamento $\wp$, denotada por $\mathbf{H}(\wp)$, como sendo

$$H(\wp) = max_{1 \leq i \leq n}(\wp^z(c_i) + z_i).$$

O **Problema do Empacotamento Tridimensional**, **PET**, pode ser formalmente definido como segue:

Dados uma caixa $B = (a, b, \infty)$ e uma lista de caixas $L = (c_1, c_2, \ldots, c_n)$, encontrar um empacotamento $\wp$ de L em B cuja altura seja a menor possível.

A notação $\mathbf{PET(a, b)}$ será usada para nos referirmos ao PET cuja caixa de entrada B é do tipo (a, b, ∞). Assim, cada instância do $PET(a, b)$ consiste simplesmente de uma lista L de caixas.

Muitas vezes a ordem em que as caixas de L aparecem não é importante. Isto justifica denotar L como um conjunto, ao invés de uma n-upla ordenada. Assim, ambas as notações serão usadas, ficando claro pelo contexto se a ordem definida é relevante ou não.

Dizemos que uma lista L é a **concatenação** de duas listas $L_1 = (c_1, c_2, \ldots, c_n)$ e $L_2 = (c'_1, c'_2, \ldots, c'_m)$, se $L = (c_1, c_2, \ldots, c_m, c'_1, c'_2, \ldots, c'_m)$. Neste caso usamos a notação $L = L_1 || L_2$.

As operações de *união*, *intersecção* e *diferença* de listas serão usadas no sentido usual, mesmo quando as listas são denotadas como n-uplas ordenadas. Nesses casos, na lista resultante a ordem das caixas é irrelevante.

Se f é uma função que assume valores reais, $f : D \rightarrow I\!R$, e S é um subconjunto (ou uma sublista) de D, $f(S)$ denota a soma $\sum_{e \in S} f(e)$.

Alguns algoritmos de empacotamento particionam a lista de caixas L em duas ou mais sublistas $L_1, L_2, \ldots, L_v$, e geram seus empacotamentos $\wp_1, \wp_2, \ldots, \wp_v$ separadamente. Depois, juntam esses empacotamentos para gerar um empacotamento da lista original L. Suponha que

$$\wp_i : L_i \rightarrow B,$$

onde $\wp_i(c) = (\wp_i^x(c), \wp_i^y(c), \wp_i^z(c))$, $c \in L_i$, $1 \leq i \leq v$. Define-se a **concatenação** (ou **combinação**) desses empacotamentos $\wp_1, \wp_2, \ldots, \wp_v$, denotado por $\wp_1 || \wp_2 || \ldots || \wp_v$, como sendo um empacotamento $\wp : L \rightarrow B$ tal que

$$\wp(c) = (\wp_i^x(c), \wp_i^y(c), \sum_{j=1}^{i-1} H(\wp_j) + \wp_i^z(c)), \quad \text{para} \quad c \in L_i, \quad 1 \leq i \leq v \ .$$

Neste caso, claramente $H(\wp) = H(\wp_1 \| \wp_2 \| \ldots \| \wp_v) = H(\wp_1) + H(\wp_2) + \cdots + H(\wp_v)$.

Vários algoritmos de empacotamento discutidos neste trabalho adotam a estratégia **nível-por-nível faixa-por-faixa**. Um empacotamento construído com esta estratégia consiste de um conjunto de níveis; cada nível consiste de um conjunto de faixas e cada faixa consiste de um conjunto de caixas.

Um **nível** em um empacotamento $\wp$ é uma região

$$N = [0, a) \times [0, b) \times [Z_1, Z_2)$$

na qual há um conjunto S de caixas tal que

$$\forall c \in S : \wp^z(c) = Z_1 \quad \text{e} \quad Z_2 - Z_1 = \max_{c \in S}(z(c)) \ .$$

Uma **faixa** em um nível N, como acima definido, é uma região

$$F = [0, a) \times [Y_1, Y_2) \times [Z_1, Z_2)$$

na qual há um conjunto S de caixas tal que

$$\forall c \in S : \wp^y(c) = Y_1 \ \text{ e } \ \wp^z(c) = Z_1 \ ; \ Y_2 - Y_1 = \max_{c \in S}(y(c)) \ \text{ e } \ Z_2 - Z_1 = \max_{c \in S}(z(c)) \ .$$

Define-se **fundo** de uma caixa c, $fundo(c)$, como sendo a região

$$fundo(c) = [0, x(c)) \times [0, y(c)).$$

Se $x(c) = y(c)$ então dizemos que a caixa c tem **fundo quadrado**.

Dada uma caixa c, denotaremos por $\mathbf{S(c)}$ a **área de fundo** da caixa c, e denotaremos por $\mathbf{V(c)}$ o **volume** de c. Mais formalmente,

$$\begin{aligned} S(c) &= x(c) \cdot y(c), \\ V(c) &= x(c) \cdot y(c) \cdot z(c). \end{aligned}$$

Neste trabalho apresentaremos algoritmos de empacotamento especialmente desenvolvidos para instâncias especiais: caixas de fundo quadrado, caixas *pequenas*, etc. Para facilitar a especificação de tais instâncias, adotaremos a seguinte notação.

- $\mathcal{R}(\mathbf{a}, \mathbf{b})$ denota o conjunto de todas as instâncias possíveis do $PET(a, b)$. Consiste de listas cujas caixas c são tais que $0 < x(c) \leq a$ e $0 < y(c) \leq b$.
- $\mathcal{Q}(\mathbf{a}, \mathbf{b})$ denota o conjunto das instâncias de $R(a, b)$ onde as caixas a serem empacotadas têm fundo quadrado.

- $\mathcal{R}_{\mathbf{m}}(\mathbf{a},\mathbf{b})$ denota as instâncias de $R(a,b)$ onde cada caixa c é tal que $x(c) \leq \frac{a}{m}$ e $y(c) \leq \frac{b}{m}$, $m \geq 1$.
- $\mathcal{Q}_{\mathbf{m}}(\mathbf{a},\mathbf{b})$ denota o conjunto $R_m(a,b) \cap Q(a,b)$.

Outras notações que serão convenientes para especificar tipos especiais de caixas são as seguintes.

$$\mathcal{X}(\mathbf{a},\mathbf{b}) = \left\{c_i = (x_i, y_i, z_i) : \ \tfrac{a}{b} < \tfrac{x_i}{y_i}\right\} \ ,$$
$$\mathcal{Y}(\mathbf{a},\mathbf{b}) = \left\{c_i = (x_i, y_i, z_i) : \ \tfrac{a}{b} \geq \tfrac{x_i}{y_i}\right\} \ ,$$
$$\mathcal{C}\,[\mathbf{p}'', \mathbf{p}' \ ; \ \mathbf{q}'', \mathbf{q}'](\mathbf{a},\mathbf{b}) = \{c_i = (x_i, y_i, z_i) : \ p''a < x_i \leq p'a \ , \quad q''b < y_i \leq q'b\} \ ,$$

onde $0 \leq p'' < p' \leq 1$, $0 \leq q'' < q' \leq 1$, $a > 0$ e $b > 0$.

1.2 Medidas de desempenho dos algoritmos

Uma abordagem utilizada para tratar problemas de otimização que são $\mathcal{NP}$-difíceis [10] é desenvolver algoritmos com complexidade polinomial que geram soluções próximas das soluções ótimas. Para analisar o desempenho de tais algoritmos, principalmente no caso de problemas de empacotamento, são usadas as medidas que definiremos a seguir.

Suponha que $\mathcal{A}$ seja um algoritmo aproximado para resolver o Problema do Empacotamento Tridimensional. Denote por $\mathcal{A}(L)$ a altura do empacotamento produzido por $\mathcal{A}$ para uma instância L, e $OPT(L)$ a altura de um empacotamento ótimo de L.

Se existe uma constante α tal que

$$\mathcal{A}(L) \leq \alpha \cdot OPT(L) \ \text{ para todo } L \ ,$$

então α é chamado um **limite de desempenho absoluto** para o algoritmo $\mathcal{A}$.

A medida mais comumente utilizada para medir o desempenho de algoritmos $\mathcal{A}$ para problemas de empacotamento é o chamado **limite de desempenho assintótico**. Para definir tal limite, supõe-se que as instâncias L consistem de caixas com altura no máximo Z e que

$$\lim_{OPT(L)\to\infty} \frac{Z}{OPT(L)} = 0.$$

Neste caso, tal limite é definido como

$$r(\mathcal{A}) = \lim_{OPT(L)\to\infty} \sup_{L} \frac{\mathcal{A}(L)}{OPT(L)}.$$

Dizemos que α é **um limite de desempenho assintótico** de um algoritmo $\mathcal{A}$ se existe uma constante β tal que para todo L, na qual toda caixa tenha altura no máximo Z, vale

$$\mathcal{A}(L) \leq \alpha \cdot OPT(L) + \beta \cdot Z \ .$$

Mais ainda, se para todo ϵ pequeno e todo N grande, ambos positivos, existe uma instância L tal que

$$\mathcal{A}(L) > (\alpha - \epsilon) \cdot OPT(L) \ \text{ e } \ OPT(L) > N,$$

então dizemos que α é um **limite justo** e neste caso temos que $r(\mathcal{A}) = \alpha$.

Se para todo $M > 1$, há uma instância L tal que

$$\mathcal{A}(L) > M \cdot OPT(L) \ ,$$

então dizemos que $\mathcal{A}$ tem um **desempenho de pior caso ilimitado**. Neste caso, claramente $\mathcal{A}$ não é um bom algoritmo.

As medidas de desempenho que apresentamos, embora definidas aqui relativamente a algoritmos para o PET, podem ser usadas para outros problemas. Utilizaremos esses conceitos para outros problemas sempre que necessário, deixando a cargo do leitor a interpretação adequada.

Capítulo 2

Algoritmos de empacotamento unidimensional e bidimensional

Neste capítulo descreveremos vários algoritmos de empacotamento unidimensional e bidimensional, principalmente aqueles que serão utilizados nos algoritmos de empacotamento tridimensional.

Na primeira seção só trataremos de algoritmos de empacotamento unidimensional. Descreveremos quatro algoritmos de empacotamento unidimensional *on-line* e um *off-line*[1].

Na segunda seção, descreveremos os algoritmos de empacotamento bidimensional. Dividimos a classe desses algoritmos em dois tipos: os de empacotamento bidimensional em faixa e os de empacotamento bidimensional em placas.

A maioria dos resultados desta seção não serão provados. Alguns dos resultados sobre empacotamento bidimensional em placas relativos a garantias de área serão provados, pois estes têm demonstrações simples, e deixarão o leitor mais familiarizado com os métodos de demonstração a serem utilizados no caso tridimensional.

Uma resenha dos principais resultados sobre os problemas de empacotamento unidimensional e bidimensional podem ser encontrados em [5].

2.1 Algoritmos de empacotamento unidimensional

O **Problema do Empacotamento Unidimensional**, **PEU**, pode ser formalmente definido como:

[1] **On-line** e **off-line** são definidos mais adiante.

Dados uma constante C e uma lista de itens $L = (p_1, p_2, \ldots, p_n)$, onde cada item p_i está associado a um valor $s(p_i)$ satisfazendo $0 \leq s(p_i) < C$, encontrar o menor inteiro m tal que L pode ser particionado em m listas $L_1, L_2, \ldots, L_m$, onde cada L_i satisfaz $s(L_i) \leq C,\ i = 1, \ldots, m$.

Um empacotamento $\mathcal{U}$ de uma lista de itens $L = (p_1, p_2, \ldots, p_n)$, satisfazendo as condições acima, é um mapeamento $(\mathcal{U}^b, \mathcal{U}^s) : (L, L) \rightarrow (\mathcal{Z}^+, [0, C))$, onde $\{L_1, L_2, \ldots, L_m\}$ é uma partição de L e $\mathcal{U}^b(p) = k$ se $p \in L_k$, e $\mathcal{U}^s(p_j^k) = \sum_{i=1}^{j-1} s(p_i^k)$, sendo $L_k = (p_1^k, p_2^k, \ldots, p_{n_k}^k)$, $k = 1, \ldots, m$.

Em geral, cada lista L_k é vista como sendo o conteúdo de uma **barra** de capacidade C, e o objetivo é minimizar o número de barras necessárias para empacotar L.

Usaremos a notação $\mathbf{PEU(C)}$ para referirmos ao problema PEU com constante de entrada C. Denotaremos por $\mathcal{A}(\mathbf{L})$ o número de barras, de comprimento C, usadas pelo algoritmo $\mathcal{A}$ para empacotar uma lista de itens L, e $\mathbf{OPT^b(L)}$ o número de barras usadas por um empacotamento ótimo de L.

Os algoritmos de empacotamento que empacotam os elementos de uma lista na ordem dada, sem precisar de informação dos elementos que ainda não foram empacotados, são chamados de algoritmos **on-line**. Os algoritmos de empacotamento que não são *on-line*, são chamados de algoritmos **off-line**.

O Problema do Empacotamento Unidimensional tem sido amplamente estudado desde o início da década de 70. Trata-se de um problema $\mathcal{NP}$-completo [10, 18], para o qual são conhecidos vários algoritmos e seus respectivos limites de desempenho assintótico [13, 14, 15, 29, 16, 1].

As provas dos resultados relativos aos limites de desempenho dos algoritmos desta seção são extensas e complicadas. Não apresentaremos aqui estas provas, pois o leitor poderá consultar diretamente nas fontes mencionadas. Citaremos aqui apenas os principais resultados.

Veremos aqui os cinco seguintes algoritmos para o PEU:

- **NF (Next Fit)**,
- **FF (First Fit)**,
- **BF (Best Fit)**,
- $\mathbf{H_M}$ **(Harmonic$_\mathbf{M}$)** e
- **FFD (First Fit Decreasing)**.

Os algoritmos NF, FF, BF e H_M são algoritmos *on-line*, enquanto o Algoritmo FFD é *off-line*.

Veremos primeiro a descrição dos algoritmos *on-line* e em seguida descrição do Algoritmo FFD.

2.1.1 Algoritmos on-line

Descreveremos primeiramente o Algoritmo NF. Antes de apresentar uma descrição formal deste algoritmo, faremos uma descrição informal. No caso dos demais algoritmos só faremos uma descrição informal.

Algoritmo NF

Dada uma lista de itens $L = (p_1, \ldots, p_n)$, o Algoritmo NF gera uma partição de L, gerando primeiro uma lista L_1, depois uma lista L_2, e assim por diante. Cada lista L_i contém os itens que podem ser empacotados em L_i. O Algoritmo NF sempre testa cada item (na ordem que ocorreu em L), verificando se o mesmo pode ser empacotado na lista corrente L_i. Enquanto isto é possível, os itens são empacotados em L_i. Quando um item p não pode ser empacotado em L_i, o Algoritmo NF pára o empacotamento em L_i e empacota p em uma nova lista L_{i+1}, que passa a ser a nova lista corrente. O algoritmo pára quando todos os itens tiverem sido empacotados, e retorna a partição $\{L_1, \ldots, L_m\}$, onde L_m é a última lista gerado pelo algoritmo.

Formalmente, o algoritmo acima pode ser descrito como segue.

Algoritmo $NF(L)$

Entrada: Lista $L = (p_1, \ldots, p_n)$ e constante C.
Saída: Partição $\{L_1, \ldots, L_m\}$ de L onde cada L_i $(1 \leq i \leq m)$ satisfaz $\sum_{p \in L_i} s(p) \leq C$.

```
% PARTICAO: Partição da lista L.
% i: Contador de sublistas de L a ser inserida na PARTICAO.
PARTICAO ← ∅.
L_1 ← ∅.
i ← 1.
para j ← 1 até n faça
   se s(L_i) + s(p_j) > C
      então  % Comecar a agrupar os itens em uma nova sublista.
         PARTICAO ← PARTICAO ∪ L_i .
         L_{i+1} ← p_j .
         i ← i + 1.
```

senão % Inserir o item na sublista corrente.
$L_i \leftarrow L_i \| p_j$.
fim se
fim para
$PARTICAO \leftarrow L_i$.
retorne $PARTICAO$.
fim algoritmo.

Algoritmo FF

Seja $\{L_1, L_2, \ldots, L_k\}$ a partição gerada pelo Algoritmo FF até um certo momento. Dada uma lista de itens $L = (p_1, \ldots, p_n)$ a ser empacotada, suponha que o Algoritmo FF tenha acabado de empacotar um item p_{j-1}. Neste ponto, para empacotar o próximo item, p_j, o algoritmo procura a lista L_i com o menor índice i tal que $s(L_i) + s(p_j) \leq C$. Caso encontre, insere p_j na lista L_i; caso contrário, insere na partição uma nova lista L_{k+1} contendo o item p_j, e repete o processo até que termine de empacotar todos os itens.

Algoritmo BF

O Algoritmo BF se assemelha ao Algoritmo FF. A diferença está no modo como a lista L_i, escolhido para receber o novo item p_j, é determinado. O Algoritmo BF procura uma lista L_i tal que $s(L_i) + s(p_j) \leq C$ e $s(L_i)$ é máximo entre as listas L_i da partição corrente. É escolhido a lista de menor índice, caso haja duas listas da partição onde as duas condições sejam satisfeitas.

Algoritmo $\mathbf{H_M}$

Para cada inteiro positivo $M \geq 2$ definimos um algoritmo denotado por H_M.

O Algoritmo H_M divide o intervalo $[0, C)$ em M subintervalos $I_k,\ k = 1, \ldots M$, $[0, C) = \cup_{k=1}^{M} I_k$, onde

$$I_k = \begin{cases} \left(\frac{C}{2}, C\right) & \text{para} \quad k = 1 \\ \left(\frac{C}{k+1}, \frac{C}{k}\right] & \text{para} \quad k = 2, \ldots, M-1 \\ \left[0, \frac{C}{M}\right] & \text{para} \quad k = M \ . \end{cases}$$

Um item p_i é chamado um I_k-*item* se $s(p_i) \in I_k,\ k = 1, \ldots, M$. O Algoritmo H_M constrói uma partição de L que é a união de M conjuntos $P_1, \ldots, P_M$. Inicialmente os conjuntos P_k são vazios, e estes vão sendo construídos à medida que os itens são empacotados. Os itens são testados na ordem dada por L, sendo que para empacotar

um item p, é determinado um inteiro k tal que p é um I_k-item, e p é empacotado em P_k usando o Algoritmo NF. O algoritmo termina quando todos os itens de L tiverem sido empacotados. Note que cada P_k é uma partição da lista formada pelos I_k-itens de L.

Limites de desempenho assintótico dos algoritmos unidimensional on-line

Trataremos aqui do $PEU(1)$. Note que isto não afeta em nada a análise dos algoritmos vistos.

Seja $S_u = \{NF, FF, BF, H_M\}$. Para analisar o desempenho dos algoritmos em S_u, são usadas as funções peso $W_{\mathcal{A}},\ \mathcal{A} \in S_u$, definidas a seguir.

- $W_{NF} : [0,1] \to [0,2]$, sendo $W_{NF}(x) = 2x$;
- $W_{FF} = W_{BF} : [0,1] \to [0,1]$,

$$\text{sendo } W_{FF}(x) = W_{BF}(x) = \begin{cases} \frac{6}{5}x & \text{se } 0 \le x \le \frac{1}{6} \\ \frac{9}{5}x - \frac{1}{10} & \text{se } \frac{1}{6} \le x \le \frac{1}{3} \\ \frac{6}{5}x + \frac{1}{10} & \text{se } \frac{1}{3} < x \le \frac{1}{2} \\ 1 & \text{se } \frac{1}{2} < x \le 1; \end{cases}$$

- $W_{H_M} : [0,1) \to [0,1]$, sendo $W_{H_M}(x) = \begin{cases} 1 & \text{se } \frac{1}{2} < x < 1 \\ \frac{1}{k} & \text{se } \frac{1}{k+1} < x \le \frac{1}{k} \\ \frac{x}{(1-\frac{1}{M})} & \text{se } 0 < x \le \frac{1}{M}. \end{cases}$

Sejam

- $U_{NF} = 2,\ \ C_{NF} = 1,\ \ D_{NF} = 1,$
- $U_{FF} = 1,7,\ \ C_{FF} = 2,\ \ D_{FF} = 8,$
- $U_{BF} = 1,7,\ \ C_{BF} = 2,\ \ D_{BF} = 8,$
- $U_{H_M} = \sum_{j=1}^{i} \frac{1}{c_j} + \frac{M}{c_{i+1}(M-1)}$, se $c_i < M \le c_{i+1}$ para algum $i \ge 1$, sendo $c_1, c_2, \ldots$ definidos como $c_1 = 1$ e $c_i = c_{i-1}(c_{i-1}+1)$ para $i \ge 2,\ C_{H_M} = M-1\ \ e\ \ D_{H_M} = 1.$

Os seguintes lemas ([9, 15, 19]) estão relacionados com as funções peso $W_{\mathcal{A}}$ e as constantes $U_{\mathcal{A}},\ C_{\mathcal{A}}\ \ e\ \ D_{\mathcal{A}}$ acima definidos.

Lema 2.1.1 *Seja* $\mathcal{A} \in S_u$ *então* $\sum_{p \in L} W_{\mathcal{A}}(s(p)) \leq U_{\mathcal{A}} \cdot OPT^b(L)$ *para toda lista de itens* L.

Lema 2.1.2 *Seja* $\mathcal{A} \in S_u$. *Então para toda lista de itens* L, *tem-se que* $\sum_{p \in L} W_{\mathcal{A}}(s(p)) \geq \mathcal{A}(L) - C_{\mathcal{A}}$.

Lema 2.1.3 *Seja* $\mathcal{A} \in S_u$. *Para todo inteiro* $N \geq 1$, *existe uma lista de itens* L *tal que* $OPT^b(L) > N$ *e* $\mathcal{A}(L) \geq U_{\mathcal{A}} \cdot OPT^b(L) - D_{\mathcal{A}}$.

O teorema a seguir reúne os lemas acima, formando um limite de desempenho assintótico para cada um dos algoritmos unidimensionais citados.

Teorema 2.1.4 *Seja* $\mathcal{A}$ *um algoritmo em* $S_u = \{NF, FF, BF, H_M\}$ *e sejam* $U_{\mathcal{A}}$ *e* $C_{\mathcal{A}}$ *como definido anteriormente. Para toda lista de itens* L, *tem-se que* $\mathcal{A}(L) \leq U_{\mathcal{A}} \cdot OPT^b(L) + C_{\mathcal{A}}$. *Mais ainda, em cada caso, o limite indicado é justo.*

Dem. Os Lemas 2.1.1 e 2.1.2 nos garantem que para toda lista de itens L, $\mathcal{A}(L) \leq U_{\mathcal{A}} \cdot OPT^b(L) + C_{\mathcal{A}}$. Pelo Lema 2.1.3, temos que U_{NF}, U_{FF} e U_{BF} são todos limites justos para o algoritmo correspondente, e U_{H_M} também é um limite justo se $M = c_i$ para algum $i \geq 2$.

□

2.1.2 Um algoritmo off-line

O algoritmo *off-line* FFD, apesar de ser bem simples, tem um limite de desempenho bastante bom. Pode ser descrito como segue.

Algoritmo FFD

Entrada: Lista $L = (p_1, \ldots, p_n)$ e constante C.
Saída: Partição $\{L_1, \ldots, L_m\}$ de L onde cada L_i $(1 \leq i \leq m)$ satisfaz $\sum_{p \in L_i} s(p) \leq C$.

1. Faça uma ordenação não-crescente da lista L.
2. Aplique o Algoritmo FF à nova lista.
3. Retorne a solução encontrada.

fim algoritmo.

Em 1973, Johnson [13] mostrou que para toda lista de itens L,

$$\mathbf{FFD(L)} \leq \frac{\mathbf{11}}{\mathbf{9}}\mathbf{OPT^b(L)} + \mathbf{4}$$

Em 1985, Baker [1] apresentou uma nova prova do limite de desempenho, melhorando o resultado para

$$\mathbf{FFD(L)} \leq \frac{\mathbf{11}}{\mathbf{9}}\mathbf{OPT^b(L)} + \mathbf{3} \ .$$

Cabe observar que o Algoritmo FFD apesar de ter um bom limite de desempenho assintótico, não é o melhor conhecido. Johnson e Garey [16] fizeram uma modificação no Algoritmo FFD obtendo um algoritmo com um limite de desempenho assintótico igual a $\frac{\mathbf{71}}{\mathbf{60}}$. Karmakar e Karp [17] desenvolveram um algoritmo que garante um empacotamento que não usa mais que $\mathbf{OPT^b(L)} + \mathbf{O(log^2 OPT^b(L))}$ barras. Para fazer empacotamentos *on-line*, Ramanam et al [28], desenvolveram um refinamento do Algoritmo H_M, que tem um limite de desempenho assintótico igual a $\mathbf{1,612}$.

Note também que o Problema do Empacotamento Unidimensional pode ser visto como um caso particular do Problema do Empacotamento Tridimensional. O problema de empacotar uma lista de itens $L = (p_1, \ldots, p_n)$ em barras de comprimento C pode ser visto como o problema de empacotar uma lista de caixas L' numa caixa B, onde $L' = (c_1, \ldots, c_n)$, $B = (1, C, \infty)$ e $c_i = (1, p_i, 1)$, para $i = 1, \ldots, n$. Claramente o empacotamento tridimensional obtido fica dividido em níveis (de altura 1), onde cada nível representa o empacotamento de itens em uma barra. A altura do empacotamento nos dá o número de barras usadas.

2.2 Algoritmos de empacotamento bidimensional

O **Problema do Empacotamento Bidimensional PEB** consiste em empacotar uma lista $L = (r_1, \ldots, r_n)$ de retângulos em um retângulo R.

Consideraremos aqui duas variantes deste problema: o Problema do Empacotamento Bidimensional em **faixa**, $\mathbf{PEB^f}$, e o Problema do Empacotamento Bidimensional em **placas**, $\mathbf{PEB^p}$. Nas próximas seções, definiremos esses dois problemas.

Um empacotamento de retângulos r_i num retângulo R é dito **ortogonal** se cada aresta de r_i é paralela a um dos lados de R. Um empacotamento ortogonal é dito **orientado** se cada retângulo r_i é representado como um par ordenado $r_i = (w(r_i), h(r_i))$, e o eixo correspondente à largura (respectivamente comprimento) de r_i é paralelo ao eixo correspondente à largura (respectivamente comprimento) de R. Ou seja, rotações de 90^0 não são permitidas. Chamaremos de **largura** e **altura** de um retângulo r os valores $w(r)$ e $h(r)$, respectivamente.

Trataremos aqui apenas do caso ortogonal e orientado, apesar do Problema do Empacotamento Bidimensional também ser estudado sem estas duas restrições. Erdös e Graham [8] mostraram que um empacotamento ortogonal de quadrados iguais em um retângulo nem sempre é melhor do que um empacotamento não ortogonal. Meir e Moser

[26] obtiveram algoritmos com garantias de empacotamento, no caso de empacotamentos de retângulos em retângulo, quando as listas não ultrapassem certo limite de área. Nestes algoritmos são efetuadas rotações de 90^0 sobre os retângulos.

2.2.1 Algoritmos de empacotamento em faixa

O **Problema do Empacotamento Bidimensional em faixa**, $\mathbf{PEB^f}$, consiste em empacotar uma lista de retângulos $L = (r_1, \ldots, r_n)$ em um retângulo $R = (a, \infty)$ de forma a minimizar a altura do empacotamento.

Denotaremos por $\mathbf{PEB^f(a)}$ o PEB^f onde $R = (a, \infty)$. Da mesma forma como fizemos para o PET, se $\mathcal{B}$ é um empacotamento de uma lista L de retângulos, denotaremos por $(\mathcal{B}^w(r), \mathcal{B}^h(r))$ as coordenadas no retângulo R onde $r \in L$ foi empacotado. Usaremos a notação $\mathcal{A}(\mathbf{L})$ e $\mathbf{OPT^f(L)}$ para indicar a altura do empacotamento gerado pelo algoritmo $\mathcal{A}$ e a altura de um empacotamento ótimo de L, respectivamente.

Note que o Problema do Empacotamento Bidimensional em faixa é um caso particular do Problema do Empacotamento Tridimensional. De fato, dada uma instância $L = (r_1, \ldots, r_n)$ do $PEB^f(a)$, basta construir uma instância $L' = (c_1, \ldots, c_n)$ do $PET(a, 1)$ tal que $c_i = (w(r_i), 1, h(r_i))$. Claramente, se $\wp$ representa um empacotamento de L' então $(\wp^x(c_i), \wp^y(c_i))$ representa as coordenadas do retângulo r_i no empacotamento de L em $R = (a, \infty)$, e $H(\wp)$ representa a altura do empacotamento bidimensional em faixa de L.

O principal algoritmo descrito nesta seção é o Algoritmo UD, mas a construção deste envolve outros algoritmos, os algoritmos $NFDH^f$ e BL. Os principais resultados relativos aos algoritmos $NFDH^f$ e BL serão enunciados mas não serão usados.

Veremos aqui os seguintes algoritmos para este problema:

- **$\mathbf{NFDH^f}$ (Next Fit Decreasing Height)**,
- **$\mathbf{FFDH^f}$ (First Fit Decreasing Height)**,
- **BL (Bottom-up Left-justified)** e
- **UD (Up-Down)**.

Descreveremos cada um deles nas seções seguintes.

Algoritmo $\mathbf{NFDH^f}$

O empacotamento gerado pelo Algoritmo $NFDH^f$ é dividido em níveis, e todos os retângulos de um nível são colocados com um lado na linha de baixo de cada nível.

Primeiramente o Algoritmo $NFDH^f$ ordena a lista $L = (r_1, \ldots, r_n)$ de forma que $h(r_1) \geq h(r_2) \geq \cdots \geq h(r_n)$. Feito isto, começa a empacotar os retângulos de L na seqüência dada pela ordenação. Para empacotar um novo retângulo r, este algoritmo tenta colocar o retângulo no último nível criado, dispondo-o mais a esquerda. Caso não consiga, cria um novo nível e coloca r neste novo nível. Neste caso, r é colocado mais à esquerda, sendo que a altura deste nível é definida pela altura de r.

O seguinte resultado foi provado por Coffman et al [6].

Teorema 2.2.1 *Para qualquer instância L do $PEB^f(a)$, onde nenhum retângulo tem altura superior a Z, tem-se que*

$$NFDH^f(L) \leq 2 \cdot OPT^f(L) + Z \ .$$

Mais ainda, este limite é justo.

Algoritmo FFDH^f

O Algoritmo $FFDH^f$ é muito parecido com o Algoritmo $NFDH^f$. De fato, ele é um melhoramento do Algoritmo $NFDH^f$. O Algoritmo $NFDH^f$ sempre que cria um novo nível, nunca mais tenta empacotar um retângulo nos níveis anteriores. Já o Algoritmo $FFDH^f$, sempre que vai empacotar um novo retângulo r, tenta empacota-lo em algum nível anteriormente criado, colocando-o no primeiro nível em que isso for possível, justificando-o sempre mais à esquerda. Caso não seja possível colocar r em nenhum dos níveis criados, então $FFDH^f$ cria um novo nível como no Algoritmo $NFDH^f$. Na Figura 2.1, exemplificamos um empacotamento feito pelo algoritmo $FFDH^f$.

Os seguintes resultados foram provados por Coffman et al [6].

Teorema 2.2.2 *Para qualquer instância L do PEB^f, onde nenhum retângulo tem altura superior a Z, tem-se que*

$$FFDH^f(L) \leq 1,7 \cdot OPT^f(L) + Z \ .$$

Ademais, este limite é justo.

Teorema 2.2.3 *Para qualquer instância L do PEB^f que consiste de uma lista de quadrados, onde nenhum retângulo tem altura superior a Z, tem-se que*

$$FFDH^f(L) \leq 1,5 \cdot OPT^f(L) + Z.$$

Ademais, este limite é justo.

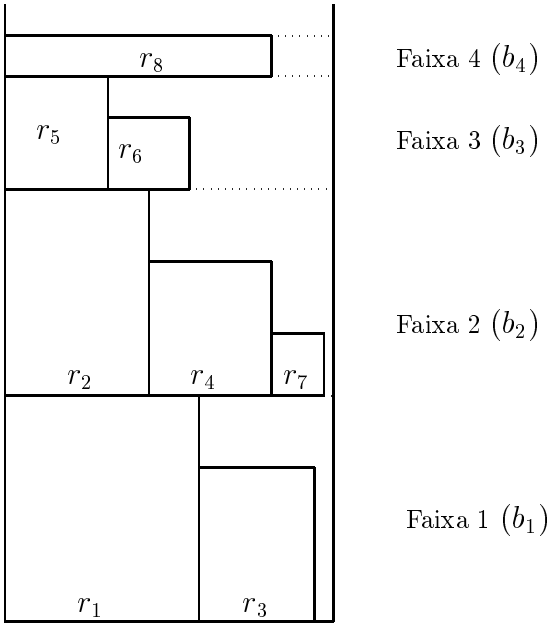


Figura 2.1: Exemplo de empacotamento gerado pelo Algoritmo $FFDH^f$.

Teorema 2.2.4 *Para qualquer instância L do $PEB^f(a)$, tal que $w(r) \leq \frac{a}{m}$, $\forall r \in L$, $m \geq 2$, tem-se que*

$$FFDH^f(L) \leq \left(\frac{m+1}{m}\right) \cdot OPT^f(L) + Z,$$

onde $Z = \max\{h(r) : \ r \in L\}$. Ademais, este limite é justo.

Algoritmo BL

O Algoritmo BL empacota os retângulos de uma lista L na ordem em que ocorrem nesta lista. Para empacotar um retângulo r, o Algoritmo BL primeiro coloca o retângulo o mais baixo possível e depois dispõe-o justificado mais à esquerda. Veja na Figura 2.2 exemplo de um empacotamento construído pelo Algoritmo BL.

Os principais resultados sobre o Algoritmo BL são devidos a Baker, Coffman e Rivest [3]. Veremos os principais deles a seguir.

Uma boa estratégia para melhorar o desempenho dos algoritmos de empacotamento, é ordenar previamente os elementos a serem empacotados (segundo uma dada dimensão). Isto foi feito, por exemplo, no Algoritmo FFD. No teorema abaixo veremos que este processo dá um bom resultado no caso do Algoritmo BL, quando $L = (r_1, \ldots, r_n)$ é ordenado tal que $w(r_1) \geq w(r_2) \geq \cdots \geq w(r_n)$.

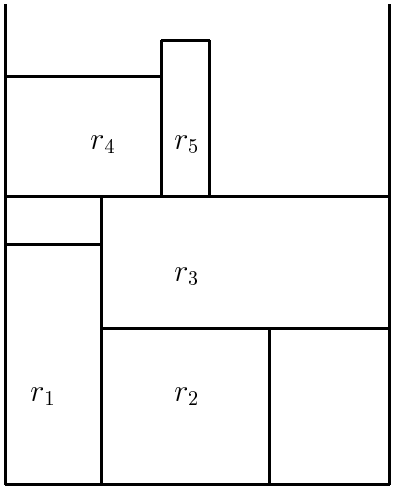


Figura 2.2: Empacotamento da lista $L = (r_1, \ldots, r_5)$ pelo Algoritmo BL.

Teorema 2.2.5 *Seja $L = (r_1, \ldots, r_n)$ uma lista de retângulos tal que $w(r_1) \geq w(r_2) \geq \cdots \geq w(r_n)$. Então*

$$BL(L) \leq 3 \cdot OPT^f(L).$$

Mais ainda, este limite é justo.

Quando L só contém quadrados, o teorema anterior pode ser melhorado.

Teorema 2.2.6 *Seja $L = (r_1, \ldots, r_n)$ uma lista de quadrados tal que $w(r_1) \geq w(r_2) \geq \cdots \geq w(r_n)$. Então*

$$BL(L) \leq 2 \cdot OPT^f(L).$$

Mais ainda, este limite é justo.

Embora este tipo de ordenação de L tenha levado a razoáveis limites de desempenho absoluto, o mesmo não é verdade para outros tipos de ordenação.

Teorema 2.2.7 *Para todo M, existe uma lista de retângulos $L = (r_1, \ldots, r_n)$ com $w(r_1) \leq w(r_2) \leq \cdots \leq w(r_n)$, $(h(r_1) \geq h(r_2) \geq \cdots \geq h(r_n))$ tal que*

$$\frac{BL(L)}{OPT^f(L)} > M.$$

Algoritmo UD

O Algoritmo UD foi desenvolvido por Baker, Brown e Katseff [2] e tem o melhor limite de desempenho assintótico conhecido para o PEB^f. Este algoritmo fez uso de três algoritmos: BL, COL (Column) e $GNFDH$ (Generalized $NFDH$).

Para que o leitor tenha uma visão geral de um empacotamento construído pelo Algoritmo UD, damos um exemplo na Figura 2.3 e a seguir, descrevemos informalmente o empacotamento gerado pelo Algoritmo UD.

Seja L uma instância do $PEB^f(a)$. O empacotamento de L gerado pelo Algoritmo UD é dividido em cinco regiões $R_1, R_2, \ldots, R_5$. Nas regiões $R_1, \ldots, R_4$, cada caixa é empacotada por um dos três algoritmos acima. Na Figura 2.3, as siglas BL, COL e N indicam que o retângulo foi empacotado pelo Algoritmo BL, COL e $GNFDH$, respectivamente. Na região R_5, todas as caixas são empacotadas pelo Algoritmo $GNFDH$. Em cada uma das regiões $R_1, \ldots, R_4$, as caixas com sigla BL são empacotadas primeiro, definindo a altura da região; depois, partindo do canto superior à direita da região, as caixas com sigla COL são colocadas uma abaixo da outra. Em seguida são as caixas com sigla N que são empacotadas pelo Algoritmo $GNFDH$ no espaço entre os empacotamentos BL e COL. Cada região $R_i, i = 1, \ldots, 5$, está definida pelas alturas h_{i-1} e h_i, como indicado na Figura 2.3. Todos os retângulos empacotados pelos algoritmos BL e COL são empacotados em ordem não-crescente de largura.

Os retângulos empacotados pelo Algoritmo BL na região $R_i, i = 1, \ldots, 4$, têm largura no intervalo $(\frac{a}{i+1}, \frac{a}{i}]$.

Seja $L = (r_1, \ldots, r_m)$ uma lista de retângulos empacotada pelo algoritmo BL tal que L está em ordem não-crescente de largura, i.e. $w(r_1) \geq w(r_2) \geq \cdots \geq w(r_m)$.

Sejam duas retas s e t paralelas ao fundo da faixa, onde L foi empacotada (veja a Figura 2.4).

Seja $d(s)$ a menor distância entre dois pontos da reta s, tal que, um dos pontos pertence à aresta direita da faixa e o outro pertence à intersecção de s com alguma aresta de um retângulo no empacotamento ou com a aresta esquerda da faixa.

Como as caixas dos empacotamentos feitos pelo Algoritmo BL na região R_i, $i = 1, \ldots, 4$, são empacotadas em ordem não-crescente de largura sendo $w(r) \in (\frac{a}{i+1}, \frac{a}{i}]$, $\forall r \in L_i$, segue que se a distância de t ao fundo da faixa é maior ou igual à distância de s ao fundo da faixa, então $d(t) \geq d(s)$. Veja a Figura 2.4.

Assim se movermos uma reta s sobre o empacotamento da região R_i, para $i = 1, \ldots, 4$, do fundo até o topo, a função $d(s)$ nos define um espaço vago que vai aumentando à medida que s se aproxima do topo do empacotamento. Este espaço é aproveitado pelo algoritmo UD, que usa o Algoritmo COL para empacotar retângulos nesta área. O Algoritmo COL empacota os retângulos um abaixo do outro, começando do canto supe-

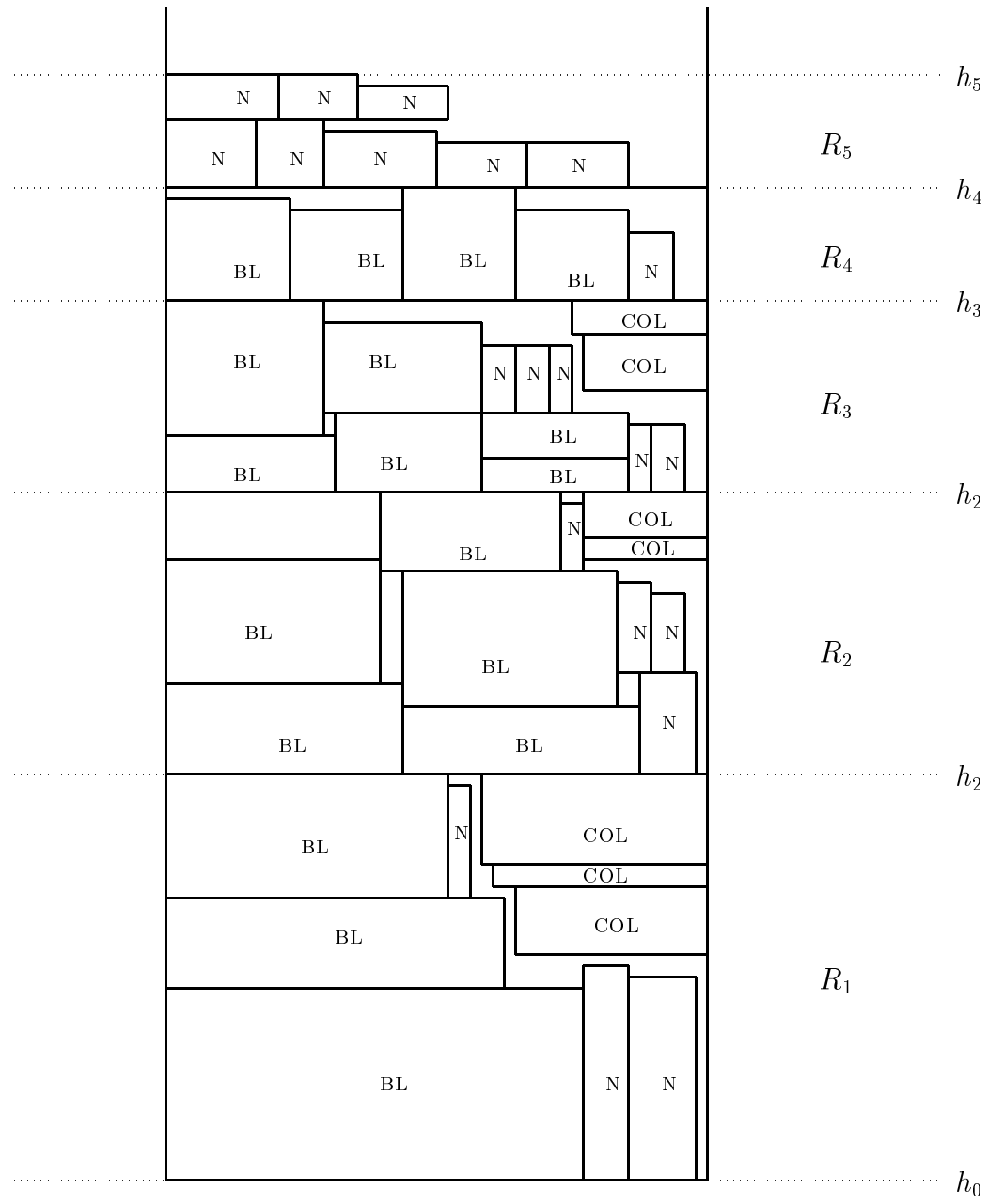


Figura 2.3: Exemplo de empacotamento gerado pelo Algoritmo UD.

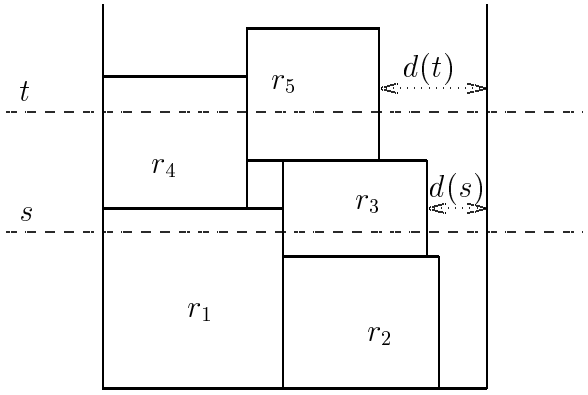


Figura 2.4: Função d sobre um empacotamento construído por BL.

rior direito da região. O nome UD (Up-Down) vem do comportamento *Bottom-Up* do Algoritmo BL e do comportamento *Top-Down* do Algoritmo COL.

Por fim, o espaço entre os empacotamentos BL e COL em cada região é aproveitado pelo Algoritmo $GNFDH$ para empacotar retângulos que têm larguras no intervalo $(0, \frac{a}{5}]$.

A região I sobre a qual o Algoritmo $GNFDH$ empacota os retângulos é irregular. Ela é delimitada superiormente e inferiormente por duas constantes $UPPER$ e $LOWER$, respectivamente, e delimitada à esquerda e direita por duas funções, $LEFT$ e $RIGHT$, respectivamente. As fronteiras definidas pelas funções $LEFT$ e $RIGHT$ são formadas por seqüências de retas horizontais e verticais, sendo que $LEFT$ e $RIGHT$ são monotonicamente crescentes.

Na Figura 2.5 mostramos um empacotamento gerado pelo Algoritmo $GNFDH$.

O primeiro retângulo r empacotado pelo Algoritmo $GNFDH$ é colocado mais ao fundo de I e justificado mais à esquerda. Com este primeiro retângulo, é definido um nível na região I, entre as alturas $LOWER$ e $LOWER + h(r)$. Os próximos retângulos são empacotados neste nível como no empacotamento $NFDH^f$, um seguido do outro, enquanto não houver sobreposição de retângulo na fronteira direita da região. Quando não se puder mais empacotar os retângulos no mesmo nível, uma nova região I' é definida, formada pela região I, delimitada superiormente e inferiormente pelas alturas $UPPER$ e $LOWER + h(r)$. Este processo se repete para a região I' com o restante dos retângulos, até que não se tenha mais retângulos para empacotar ou quando não puder ser criado um novo nível.

Agora podemos descrever o Algoritmo UD mais detalhadamente.

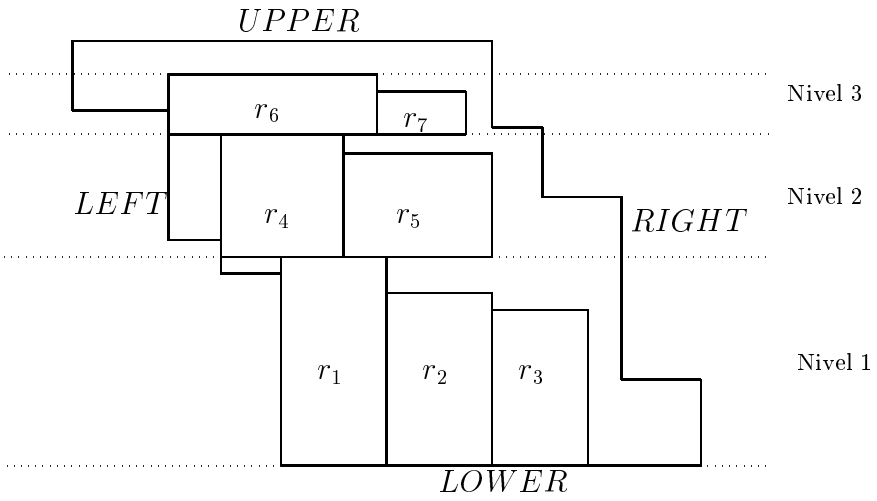


Figura 2.5: Exemplo de empacotamento gerado pelo Algoritmo $GNFDH$.

Primeiro UD divide a lista L em cinco sublistas, $L_1, \ldots, L_5$, tal que

$$\begin{aligned} L_i &= \left\{ r \in L : w(r) \in \left(\frac{a}{i+1}, \frac{a}{i} \right] \right\}, \quad \text{para} \quad i = 1, \ldots, 4 \\ L_5 &= \left\{ r \in L : w(r) \in \left(0, \frac{a}{5} \right] \right\}, \end{aligned}$$

e então ordena as listas L_i, $(i = 1, \ldots, 4)$ em ordem não-crescente de largura, e a lista L_5 em ordem não-crescente de altura.

As listas $L_1, \ldots, L_5$ são empacotadas nesta ordem, e os retângulos de L_i são empacotadas na ordem dada por L_i, $i = 1, \ldots, 5$.

Para empacotar um retângulo r de uma lista L_i, $i = 1, \ldots, 4$, o algoritmo UD tenta empacotar r nas regiões $R_1, \ldots, R_{i-1}$ (nesta ordem), pelo Algoritmo COL. Caso não consiga, empacota na região R_i usando o Algoritmo BL.

Para empacotar os retângulos de L_5, note que a função d anteriormente descrita, juntamente com a forma do empacotamento gerado pelo Algoritmo COL nos define um espaço E_i entre os empacotamentos BL e COL em cada região R_i, $i = 1, \ldots, 4$. O Algoritmo UD empacota os retângulos de L_5 nos espaços $E_1, \ldots, E_4$, tentando empacotar os retângulos nesta ordem, usando o Algoritmo $GNFDH$.

Caso ainda restem retângulos de L_5 que ainda não foram empacotados, estes são empacotados pelo Algoritmo $GNFDH$ acima da altura h_4, definindo a região R_5.

Baker, Brown e Katseff [2] provaram o seguinte resultado.

Teorema 2.2.8 *Para qualquer lista L de retângulos de altura no máximo Z, temos que*

$$UD(L) \leq \frac{5}{4} OPT^f(L) + \frac{53}{8} Z \ .$$

Mais ainda, o limite de desempenho assintótico $\frac{5}{4}$ é justo.

2.2.2 Algoritmos de empacotamento bidimensional em placas

O **Problema do Empacotamento Bidimensional em Placas**, $\mathbf{PEB^p}$, consiste em empacotar uma lista de retângulos $L = (r_1, \ldots, r_n)$ em retângulos (ou placas) $R = (a, b)$, de forma a minimizar o número de placas usadas.

Denotaremos por $\mathbf{PEB^p(a, b)}$ o Problema do Empacotamento Bidimensional em placas quando as placas R são da forma $R = (a, b)$. Denotaremos por $\mathbf{OPT^p(L)}$ o menor número de placas R necessárias para empacotar L, e $\mathcal{A}(\mathbf{L})$ o número de placas R usadas pelo algoritmo $\mathcal{A}$ para empacotar L.

Um empacotamento $\mathcal{B}$ de L em R, além de dar a coordenada na qual cada retângulo r foi empacotado, também deve dar a placa em que r foi empacotado. Denotaremos por $\mathcal{B}^{\mathbf{p}}(\mathbf{r})$ a placa onde r foi empacotada e por $(\mathcal{B}^{\mathbf{w}}(\mathbf{r}), \mathcal{B}^{\mathbf{h}}(\mathbf{r}))$ a coordenada no retângulo $\mathcal{B}^p(r)$, onde r foi empacotada.

Veremos aqui quatro algoritmos para o **Problema do Empacotamento Bidimensional em Placas**, $\mathbf{PEB^p}$:

- $\mathbf{NFDH^p}$ **(Next Fit Decreasing Height)**,
- $\mathbf{NFDW^p}$ **(Next Fit Decreasing Width)**,
- $\mathbf{MNFDH^p}$ **(Modified** $\mathbf{NFDH^p}$**)** e
- $\mathbf{MNFDW^p}$ **(Modified** $\mathbf{NFDW^p}$**)**.

Descreveremos cada um desses algoritmos e apresentaremos alguns resultados. Veremos também um exemplo de como os resultados sobre a área dos retângulos podem ser usados para se achar limites de desempenho assintótico de algoritmos para o PEB^p.

Usando garantias de área mínima

Algoritmo $\mathbf{NFDH^p}$

O Algoritmo $NFDH^p$ ordena a lista de retângulos L de forma que $L = (r_1, \ldots, r_n)$ e $h(r_1) \geq h(r_2) \geq \cdots \geq h(r_n)$. Define uma faixa de altura $h(r_1)$ paralela ao eixo w,

e empacota os retângulos seguidamente na faixa criada, até que um retângulo r_i não possa ser empacotado nesta faixa. Neste momento, uma nova faixa de altura $h(r_i)$ é criada, e o empacotamento continua nesta faixa. Este processo continua até que uma nova faixa não possa ser criada na mesma placa. Neste instante, $NFDH^p$ começa o empacotamento em uma nova placa, repetindo o processo até que todos os retângulos tenham sido empacotados. Uma descrição mais formal deste algoritmo é dada a seguir.

Algoritmo NFDH$^{\mathrm{p}}$(L)

Entrada: Lista de retângulos L e constantes a e b.
Saída: Empacotamento $\mathcal{B}$ de L em retângulos $R(a, b)$.

% $PLACA$: número da placa a receber o próximo retângulo.
% i: contador do número do retângulo sendo empacotado.
% alt_faixa: altura da faixa corrente na placa.
% h_faixa: altura da faixa.
Ordene L **obtendo** $L = (r_1, \ldots, r_n)$ **com** $h(r_i) \geq h(r_{i+1}),\ i = 1, \ldots, n-1$**.**
$PLACA \leftarrow 0$ **.**
$i \leftarrow 1$ **.**
enquanto $(i < n)$ **faça** % Loop para empacotar todas as caixas.
  $PLACA \leftarrow PLACA + 1$ **.**
  $alt\_faixa \leftarrow 0$ **.**
  $h\_faixa \leftarrow h(r_i)$ **.**
  % Loop para empacotar as caixas nos níveis.
  enquanto $(alt\_faixa + h\_faixa \leq b)$ **e** $(i \leq n)$ **faça**
    $cmp\_faixa \leftarrow 0$ **.**
    % Loop para empacotar as caixas em faixas.
    enquanto $(cmp\_faixa + w(r_i) \leq a)$ **e** $(i \leq n)$ **faça**
      $(\mathcal{B}^p(r_i), \mathcal{B}^h(r_i), \mathcal{B}^w(r_i)) \leftarrow (PLACA, alt\_faixa, cmp\_faixa)$ **.**
      $cmp\_faixa \leftarrow cmp\_faixa + w(r_i)$ **.**
      $i \leftarrow i + 1$ **.**
    fim enquanto .
    $alt\_faixa \leftarrow alt\_faixa + h\_faixa$ **.**
    $h\_faixa \leftarrow h(r_i)$ **.**
  fim enquanto .
fim enquanto .
fim algoritmo.

Algoritmo MNFDH^p

O Algoritmo $MNFDH^p$ é muito parecido com o Algoritmo $NFDH^p$. A diferença é que cada um dos retângulos com largura superior a $\frac{a}{2}$ é empacotado em uma única faixa. Com relação aos retângulos restantes, o algoritmos se comporta como o $NFDH^p$.

Note que se L é uma lista de retângulos que pode ser empacotada em uma placa R pelo Algoritmo $MNFDH^p$, então L também pode ser empacotada na placa R pelo Algoritmo $NFDH^p$.

Meir e Moser [26] conseguiram vários resultados relacionando a área de uma lista de retângulos L com a capacidade de os algoritmos $NFDH^p$ e $MNFDH^p$ empacotarem L em um único retângulo (a, b). Mencionamos alguns deles a seguir.

Teorema 2.2.9 *Qualquer lista de quadrados de lados $x_1, x_2, \ldots, x_n$ com área total A pode ser empacotado pelo Algoritmo $NFDH^p$ em qualquer retângulo (a, b) se*
(C1) $a > \overline{x}$;
(C2) $b > \overline{x}$;
(C3) $\overline{x}^2 + (a - \overline{x})(b - \overline{x}) \geq A$;
onde $\overline{x} = \max\{x_i : i = 1, \ldots, n\}$. Em alguns casos este resultado é o melhor possível.

Dem. Sem perda de generalidade, considere $x_1 \geq x_2 \geq \cdots \geq x_n$. Ao aplicar o Algoritmo $NFDH^p$ para fazer um empacotamento num retângulo de largura a, são criadas faixas, digamos $F_1, F_2, \ldots, F_v$ (nesta ordem), onde

$$F_i = ((x_{k_i}, x_{k_i}), (x_{k_i+1}, x_{k_i+1}), \ldots, (x_{k_{i+1}-1}, x_{k_{i+1}-1})),$$

e os inteiros k_f são definidos como $k_1 = 1$ e

$$a - x_{k_{f+1}} < \sum_{j=k_f}^{k_{f+1}-1} x_j \leq a, f = 1, 2, \ldots, v-1, \tag{2.1}$$

como ilustrado na Figura 2.6.

Note que para toda área A_f ocupada por quadrados na f-ésima faixa, temos por (2.1),

$$A_f = \sum_{j=k_f}^{k_{f+1}-1} x_j^2 \geq x_{k_f}^2 + (a - x_{k_f})x_{k_{f+1}} - x_{k_{f+1}}^2, f = 1, 2, \ldots, v. \tag{2.2}$$

Para efeito de cálculo, considere que $x_{k_{v+1}} = 0$.

De (2.2) obtemos para a área total A,

$$A = \sum_{f=1}^{v} A_f$$

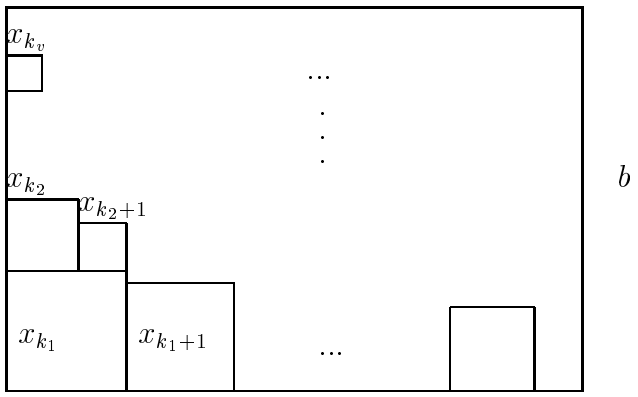


Figura 2.6: Empacotamento gerado pelo Algoritmo $NFDH^p$

$$\begin{aligned} &\geq x_1^2 + \sum_{f=1}^{v}[(a - x_{k_f})x_{k_{f+1}}] - x_{k_{v+1}} \\ &\geq x_1^2 + (a - x_{k_1})\sum_{f=1}^{v} x_{k_{f+1}} \\ &= x_1^2 + (a - x_{k_1})\sum_{f=2}^{v} x_{k_f}. \end{aligned} \tag{2.3}$$

Por hipótese, $x_1^2 + (a - x_1)(b - x_1) \geq A$. Assim, de (2.3) segue que

$$x_1 + \sum_{f=2}^{v} x_{k_f} \leq b,$$

ou seja,

$$\sum_{f=1}^{v} x_{k_f} \leq b.$$

Deste último fato concluimos que o Algoritmo $NFDH^p$ empacota os quadrados num retângulo (a, b).

Para mostrar que este resultado é o melhor possível em certos casos, considere uma lista de quadrados de lados $x_1 = x_2 = \ldots = x_{n^2+1} = \frac{1}{\sqrt{n^2+1}}$, para n inteiro e $a = b = \frac{n^2+1-\epsilon}{\sqrt{n^2+1}}$. Note que à medida que ϵ tende a zero, o lado esquerdo da condição *(C3)* se aproxima do lado direito.

□

Um dos problemas que torna difícil a utilização do resultado do Teorema 2.2.9, é que ele depende do comprimento do maior quadrado. O corolário seguinte troca esta dependência por um limite envolvendo os valores a, b e um inteiro $m \geq 2$, fazendo com que uma lista L de retângulos possa ser empacotada em um retângulo (a, b), caso a área de L não ultrapasse uma certa razão de (a, b).

Corolário 2.2.10 *Qualquer lista de quadrados de lados $x_1, x_2, \ldots, x_n$, com área total A, pode ser empacotado pelo Algoritmo $NFDH^p$ em qualquer retângulo (a, b) se $m \geq 2$*
(C1) $\overline{x} \leq \frac{a}{m}$;
(C2) $\overline{x} \leq \frac{b}{m}$;
(C3) $A \leq ab(\frac{1}{m^2} + (1 - \frac{1}{m})^2)$;
onde $\overline{x} = \max\{x_i : i = 1, \ldots, n\}$.

Dem. Para isto basta provar que

$$ab\left(\frac{1}{m^2} + \left(1 - \frac{1}{m}\right)^2\right) \leq \overline{x}^2 + (a - \overline{x})\,(b - \overline{x})\,,$$

e aplicar o Teorema 2.2.9. A desigualdade acima é provada mostrando que a função $f(x, y) = xy + (a - x)(b - y)$ tem ponto de mínimo em $(\frac{a}{2}, \frac{b}{2})$.

□

O teorema seguinte mostra que a área necessária para empacotar uma lista de quadrados em um retângulo não é maior que o dobro da área dos quadrados a serem empacotados.

Teorema 2.2.11 *Qualquer lista de quadrados de lados $x_1, x_2, \ldots, x_n$, com área total A, pode ser empacotado pelo Algoritmo $MNFDH^p$ em um retângulo (a, b), se*
(C1) $a \geq \overline{x}$;
(C2) $b \geq \overline{x}$;
(C3) $A \leq ab/2$;
onde $\overline{x} = \max\{x_i : i = 1, \ldots, n\}$. Em alguns casos, este resultado é o melhor possível.

Dem. Suponha, sem perda de generalidade, que $x_1 \geq x_2 \geq \cdots \geq x_p > \frac{a}{2} \geq x_{p+1} \geq \cdots \geq x_n$.

Sejam $F_1, F_2, \ldots, F_v$ as faixas criadas pelo Algoritmo $MNFDH^p$ nesta ordem, onde $F_i = ((x_{k_i}, x_{k_i}), (x_{k_i+1}, x_{k_i+1}), \ldots, (x_{k_n}, x_{k_n}))$, e os inteiros k_f $(f = 1, 2, \ldots, v)$ são definidos como na demonstração do Teorema 2.2.9. Então,

$$A_f = x_f^2 \geq \frac{a}{2} x_f, \quad \text{para} \quad f = 1, 2, \ldots, p - 1, \tag{2.4}$$

e por (2.2) temos que

$$A_f \geq x_{k_f}^2 - x_{k_{f+1}}^2 + \frac{a}{2} x_{k_{f+1}}, \quad \text{para} \quad f = p+1, \ldots, v, \tag{2.5}$$

onde $x_{v+1} = 0$.

Para $f = p$ temos

$$\begin{aligned} A_p &\geq x_p^2 - x_{k_{p+1}}^2 + (a - x_p)x_{p+1} \\ &= -x_{k_{p+1}}^2 + \frac{a}{2}(x_p + x_{k_{p+1}}) + (x_p - \frac{a}{2})(x_p - x_{k_{p+1}}) \\ &\geq -x_{k_{p+1}}^2 + \frac{a}{2} x_p + \frac{a}{2} x_{k_{p+1}}. \end{aligned} \tag{2.6}$$

De (2.4),(2.5) e (2.6), seque que

$$A = \sum_{f=1}^{v} A_f \geq \frac{a}{2}(\sum_{f=1}^{p} x_f + \sum_{f=p}^{v} x_{k_v}) = \frac{a}{2} \sum_{f=1}^{v} x_{k_f}. \tag{2.7}$$

Por hipótese, $\frac{ab}{2} \geq A$. Este fato juntamente com (2.7) implica que

$$b \geq \sum_{f=1}^{v} x_{k_v},$$

mostrando assim que a lista de quadrados pode ser empacotada num retângulo (a, b).

Para mostrar que este resultado é o melhor possível, considere dois quadrados, um com lado $\frac{1}{2}$ e outro com lado $\frac{1}{2} + \delta$, onde $0 < \delta < 1$. Tais quadrados não podem ser empacotados em um quadrado unitário, mas a soma de suas áreas é $\frac{1}{2} + \delta + \delta^2$, e este valor pode se tornar tão próximo de $\frac{1}{2}$ quanto se queira. Assim, se a área dos quadrados dados for só um pouco maior que $\frac{ab}{2}$, então já não podemos garantir que $MNFDH^p$ empacota tais quadrados.

Isto completa a demonstração.

□

No caso de empacotamento de retângulos, Meir e Moser [26] conseguiram o seguinte resultado.

Teorema 2.2.12 *Qualquer lista de retângulos* $L = ((x_1, y_1), \ldots, (x_n, y_n))$ *e área total* A, *pode ser empacotada pelo Algoritmo* $MNFDH^p$ *em um retângulo* (a, b) *se*
(C1) $x_i \geq y_i$, para $i = 1, \ldots, n$;
(C2) $y_i \geq y_{i+1}$, para $i = 1, \ldots, n-1$;
(C3) $a \geq \max\{x_i : i = 1, \ldots, n\}$;
(C4) $b \geq \max\{y_i : i = 1, \ldots, n\}$;
(C5) $A \leq \frac{a}{16}(8b - a)$.
Mais ainda, todas estas condições são necessárias para garantir o empacotamento de L.

Dem. Sejam $F_1, F_2, \ldots, F_v$ as faixas criadas pelo Algoritmo $MNFDH^p$ nesta ordem, onde $F_i = ((x_{k_i}, x_{k_i}), (x_{k_i+1}, x_{k_i+1}), \ldots, (x_{k_{i+1}-1}, x_{k_{i+1}-1}))$. Defina $k_f, f = 1, 2, \ldots, v$ da mesma forma que no Teorema 2.2.9.

Suponha, sem perda de generalidade, que $x_i > \frac{a}{2}$, para $i = 1, \ldots, p$ e $x_i \leq \frac{a}{2}$, para $i = p+1, \ldots, n$. Assim, temos que

$$\begin{aligned} F_f &= ((x_f, y_f)), \text{ para } f = 1, \ldots, p \text{ e} \\ F_f &= ((x_{k_f}, y_{k_f}), (x_{k_f+1}, y_{k_f+1}), \ldots, (x_{k_{f+1}-1}, y_{k_{f+1}-1})), \text{ para } f = p+1, \ldots, v, \end{aligned}$$

sendo $k_{p+1} = p + 1$ e $x_{v+1} = 0$.

Os valores de k_f, para $f = p+1, \ldots, v$, são definidos analogamente como na equação (2.1). Neste caso, temos que

$$A_f = x_f y_f \geq \frac{a}{2} y_f, \text{ para } f = 1, \ldots, p, \tag{2.8}$$

e para as faixas restantes temos, da mesma forma que em (2.2),

$$A_f > x_{k_f} y_{k_f} + (a - x_{k_f}) y_{k_{f+1}} - x_{k_{f+1}} y_{k_{f+1}}, \text{ para } f = p+1, \ldots, v, \tag{2.9}$$

onde $x_{k_{v+1}} = 0$ e $y_{k_{v+1}} = 0$. De (2.8) e (2.9) temos que

$$\begin{aligned} A &= \sum_{f=1}^{v} A_f \geq \frac{a}{2} \sum_{f=1}^{p} y_f + x_{k_1} y_{k_1} + \sum_{f=p+1}^{v-1} (a - x_{k_f}) y_{k_{f+1}} \\ &\geq \frac{a}{2} \sum f = 1 y_f + y_{k_1}^2 + \frac{a}{2} \sum_{f=p+1}^{v-1} y_{k_{f+1}}, \end{aligned}$$

desde que $x_{k_1} \geq y_{k_1}$ e $(a - x_{k_f}) \geq \frac{a}{2}$.

Assim,

$$A - y_{k_1}^2 + \frac{a}{2} y_{k_1} \geq \frac{a}{2} (\sum_{f=1}^{p} y_f + \sum_{f=p+1}^{v} y_{k_f}). \tag{2.10}$$

Agora, observe que $\frac{a}{2}y_{k_1} - y_{k_1}^2 \leq \frac{a^2}{16}$ para todo $0 \leq y_{k_1} \leq \frac{a}{2}$. Então

$$\frac{a^2}{16} + A \geq \frac{a}{2}\sum_{f=1}^{v} y_{k_f} \ ,$$

i.e.,

$$\frac{a^2}{8} + 2A \geq a\sum_{f=1}^{v} y_{k_f}.$$

Por hipótese, $\frac{1}{a}(2A + \frac{a^2}{8}) \leq ab$. Assim, concluimos que

$$b \geq \sum_{f=1}^{v} y_{k_f},$$

o que mostra que a lista de retângulos pode ser empacotada no retângulo (a, b). Para completar a demonstração, provaremos a seguir a necessidade de cada uma das condições do teorema.

Caso 1: A condição $x_i \geq y_i$ é violada e as outras condições são satisfeitas. Considere $L = ((\frac{\varepsilon}{4}, 1), (1, \frac{\varepsilon}{4})$, com $\varepsilon < \frac{7}{8}$. Claramente, L não pode ser empacotada por $MNFDH^p$.

Caso 2: A condição $y_1 \geq y_2 \geq \cdots \geq y_n$ é violada e as outras condições são satisfeitas. Considere $L = ((\frac{1}{3}, \varepsilon), (\frac{1}{3}, \varepsilon), (\frac{1}{3}, \frac{1}{3}), (\frac{1}{3}, \varepsilon), (\frac{1}{3}, \varepsilon), (\frac{1}{3}, \frac{1}{3}), (\frac{1}{3}, \varepsilon), (\frac{1}{3}, \varepsilon), (\frac{1}{3}, \frac{1}{3}), (\frac{1}{3}, \varepsilon))$, onde $0 < \varepsilon \leq \frac{5}{512}$. Claramente, $x_i \geq y_i$ e a soma das áreas é $\frac{7}{3}\varepsilon + \frac{1}{3}$ que é menor que $\frac{7}{16}$. Entretanto, L não pode ser empacotada por $MNFDH^p$.

Caso 3: A condição $\sum_{i=1}^{n} x_i y_i \leq \frac{7}{16}$ é violada e as demais condições são satisfeitas. Considere $L = ((\frac{1}{2} + \delta, \frac{1}{4} + \delta), (\frac{1}{2} + \delta, \frac{1}{4} + \delta), (\frac{1}{2} + \delta, \frac{1}{4} + \delta), (\frac{1}{4} - 2\delta, \frac{1}{4} - 2\delta))$, Note que $\sum_{i=1}^{n} x_i y_i = \frac{7}{16} + \frac{5}{4}\delta + 7\delta^2 \geq \frac{7}{16}$. Claramente, L não pode ser empacotada por $MNFDH^p$.

□

O corolário seguinte é simplesmente um caso especial do Lema 2.2.12. Note que como se trata de um empacotamento de retângulos em um retângulo R, e não é permitido rotações dos retângulos, então R pode ser reparametrizado em um quadrado $(1, 1)$, sendo que os retângulos sofrem reparametrização proporcional.

Corolário 2.2.13 *Uma lista de retângulos $L = ((x_1, y_1), \ldots, (x_n, y_n))$ pode ser empacotada em um quadrado unitário pelo Algoritmo $MNFDH^p$ se*
(C1) $x_i \geq y_i$, para $i = 1, \ldots, n$;
(C2) $y_1 \geq y_2 \geq \cdots \geq y_n$;
(C3) $\sum_{i=1}^{n} x_i y_i \leq \frac{7}{16}$.

□

Note que sob as condições *(C1)* e *(C2)*, o Corolário 2.2.13 mostra que a área necessária para empacotar uma lista de retângulos de área A em um quadrado, não é maior que $\frac{16}{7}A$.

Quando os retângulos em $L = ((x_1, y_1), \ldots, (x_n, y_n))$ são tais que $x_i \leq \frac{a}{m}$ e $y_i \leq \frac{b}{m}$, $m \geq 3$, para $i = 1, \ldots, n$, então o Algoritmo $MNFDH^p$ pode nos dar resultados melhores, como foi provado por Li e Cheng em [23].

Teorema 2.2.14 *Uma lista de retângulos $L = ((x_1, y_1), \ldots, (x_n, y_n))$ pode ser empacotada em um retângulo (a, b) pelo Algoritmo $MNFDH^p$, se existe um inteiro $m \geq 3$ tal que*
(C1) $x_i \leq \frac{a}{m}$;
(C2) $y_i \leq \frac{b}{m}$;
(C3) $\sum_{i=1}^{n} x_i y_i \leq (1 - \frac{1}{m})^2 ab$.

Dem. Suponha que haja t faixas no empacotamento, e que $x_i \geq x_{i+1}$, para $i = 1, \ldots, n-1$. Sejam x_{k_i} e y_{k_i} o comprimento e a largura do primeiro retângulo na i-ésima faixa, respectivamente; considere $k_1 = 1$.

Seja A_i a área total de todos os retângulos da i-ésima faixa, para $i = 1, \ldots, t$; e A a área acumulativa de todos os retângulos.

Desde que a largura de todos os retângulos na i-ésima faixa não são menores que $y_{k_{i+1}}$, e o retângulo $(x_{k_{i+1}}, y_{k_{i+1}})$ não pode ser empacotado na i-ésima faixa, temos as seguintes relações:

$$\begin{aligned}
A_1 &\geq x_{k_1} y_{k_1} + (a - x_{k_1}) y_{k_2} - x_{k_2} y_{k_2}, \\
A_2 &\geq x_{k_2} y_{k_2} + (a - x_{k_2}) y_{k_3} - x_{k_3} y_{k_3}, \\
&\cdots \\
A_{t-1} &\geq x_{k_{t-1}} y_{k_{t-1}} + (a - x_{k_{t-1}}) y_{k_t} - x_{k_t} y_{k_t} \quad \text{e} \\
A_t &\geq x_{k_t} y_{k_t}.
\end{aligned}$$

Somando as desigualdades acima temos que

$$A = \sum_{i=1}^{t} A_i \tag{2.11}$$

$$\geq x_{k_1} y_{k_1} + \sum_{i=1}^{t-1} (a - x_{k_i}) y_{k_{i+1}} \tag{2.12}$$

$$\geq x_{k_1} y_{k_1} + (1 - \frac{1}{m}) a \sum_{i=2}^{t} y_{k_i}. \tag{2.13}$$

Pelas hipóteses assumidas, segue que

$$A \leq (1 - \frac{1}{m})^2 ab \tag{2.14}$$

$$= \quad (1 - \frac{1}{m})a(b - \frac{b}{m}) \tag{2.15}$$

$$\leq \quad (1 - \frac{1}{m})a(b - y_1). \tag{2.16}$$

Como $x_1 y_1 > 0$, de (2.11) e (2.14), concluimos que

$$b - y_1 > \sum_{i=2}^{t} y_{k_i} \ ,$$

i.e.,

$$b > \sum_{i=1}^{t} y_{k_i} \ ,$$

o que completa a demonstração. □

Note que todos os algoritmos de empacotamento bidimensional em placas que mencionamos empacotam os retângulos usando uma ordenação não-crescente da altura destes. É claro que existem algoritmos análogos que usam uma ordenação não-crescente sobre a largura, e todos estes resultados também são válidos. Denotaremos por **NFDW^p (Next Fit Decreasing Width)** e **MNFDW^p (Modified NFDW^p)**, os algoritmos análogos aos algoritmos $NFDH^p$ e $MNFDH^p$, com ordenação não-crescente sobre a largura e empacotamento na direção da largura.

Veremos agora como a garantia de área sobre cada placa pode ser usada para se achar um limite de desempenho assintótico, de um algoritmo para o PEB^p.

Dada uma lista L, suponha que exista um algoritmo $\mathcal{A}$, para o PEB^p, que garante pelo menos $f(a,b)$ de área ocupada em cada placa (a,b), exceto talvez na última placa.

Claramente, $OPT^p(L) \geq \left\lceil \frac{S(L)}{ab} \right\rceil$. Como $S(L) \geq (\mathcal{A}(L) - 1) \cdot f(a,b)$, temos que

$$\mathcal{A}(L) \leq \frac{ab}{f(a,b)} OPT^p(L) + 1.$$

Podemos também criar novos algoritmos para o PEB^p, usando as características de empacotamento gerado pelo Algoritmo $NFDH^p$.

Vimos no Corolário 2.2.13 que uma lista L, tal que $S(L) \leq \frac{7}{16}$, pode ser empacotada pelo Algoritmo $NFDH^p$ em um quadrado unitário. Defina um novo algoritmo $\mathcal{A}$ tal que este separa todas as caixas com área maior que $\frac{7}{32}$ e empacota estas em apenas uma placa. Em seguida, $\mathcal{A}$ aplica o Algoritmo $NFDH^p$ para empacotar as caixas restantes. Note que há pelo menos $\mathcal{A}(L) - 1$ placas contendo retângulos somando área maior que $\frac{7}{32}$. Como visto anteriormente, vale que

$$\mathcal{A}(L) \leq \frac{32}{7} OPT^p(L) + 1.$$

Capítulo 3

Algoritmos para o Problema do Empacotamento Tridimensional

Muitos algoritmos para o caso bidimensional foram construídos a partir de generalizações de outros existentes no caso unidimensional. Da mesma forma, Li e Cheng [22] desenvolveram algoritmos para o Problema do Empacotamento Tridimensional (PET), generalizando algoritmos de empacotamento bidimensional em faixa. Apresentaremos estes algoritmos na Seção 3.1 deste capítulo.

Uma técnica bastante usada na análise de desempenho de algoritmos de empacotamento bidimensional em faixa é a de relacionar a altura do empacotamento com a área ocupada. Esta técnica, também pode ser generalizada para o caso tridimensional, desta vez relacionando a altura do empacotamento com o volume ocupado. No mesmo artigo, acima citado, Li e Cheng usaram esta técnica para o PET, analisando algoritmos gerais bem como algoritmos especializados para os casos em que o fundo das caixas é *pequeno* ou quadrado. Nesta mesma linha, aperfeiçoamos esta técnica e conseguimos provar que alguns algoritmos que desenvolvemos têm limite de desempenho assintótico melhores que aqueles apresentados em [22]. Tais algoritmos serão apresentados na Seção 3.2.

Recentemente, Li e Cheng [25] generalizaram algumas idéias presentes nos algoritmos de empacotamento unidimensional (vistos no Capítulo 2) e obtiveram uma classe de algoritmos com bons limites de desempenho assintótico. Entretanto, estes algoritmos possuem constante aditiva muito alta. Uma comparação entre o limite de desempenho assintótico destes algoritmos e de outro desenvolvido por nós será feita no fim deste capítulo.

Na descrição de cada algoritmo procuramos deixar claro para quais tipos de instâncias o algoritmo se aplica. Como veremos, em geral, os algoritmos recebem como entrada uma lista L de caixas e duas constantes a e b. Estas constantes correspondem à largura e o comprimento da caixa $B = (a, b, \infty)$ onde a lista L deve ser empacotada. Quando porém, fazemos uso de tais algoritmos na descrição de outros, fazemos a chamada passando como

parâmetro apenas uma lista. Assim, quando um determinado Algoritmo $\mathcal{A}$ é chamado, em geral esta chamada é indicada simplesmente por $\mathcal{A}(L)$. A rigor deveríamos especificar os parâmetros a e b, pois muitas vezes fazemos uma parametrização e queremos que esses valores sejam, por exemplo, iguais a 1. Para simplificar a notação, não faremos isto. Em geral, o contexto deixará claro a quais valores estamos nos referindo.

3.1 Generalizações dos algoritmos $NFDH^f$ e $FFDH^f$

Vimos no capítulo anterior, os Algoritmos $NFDH^f$ e $FFDH^f$ para o Problema do Empacotamento Bidimensional em faixa. Veremos nesta seção dois algoritmos para o Problema do Empacotamento Tridimensional, que são uma generalização desses algoritmos. Indicaremos esses algoritmos pelos seguintes nomes mnemônicos.

- **NFDH (Next Fit Decreasing Height)** e
- **FFDH (First Fit Decreasing Height)**.

Li e Cheng [22], mostraram que estes algoritmos podem não levar a bons empacotamentos, podendo até mesmo ter desempenho de pior caso ilimitado. Estes resultados serão vistos nas seções seguintes.

3.1.1 Algoritmo $NFDH$

Descreveremos nesta seção duas variantes para o Algoritmo $NFDH$, denominadas $NFDH^x$ e $NFDH^y$. O nome $NFDH$ será usado para nos referirmos indistintamente a qualquer uma dessas variantes.

Antes de descrevermos o Algoritmo $NFDH^x$, veremos um algoritmo simples que empacota as caixas nível-por-nível, faixa-por-faixa, denominado de **NF (Next Fit)**. Este Algoritmo também possui duas variantes a serem denominadas NF^x e NF^y. O Algoritmo NF é usado pelo Algoritmo $NFDH$. Faremos primeiramente uma descrição informal, e depois, uma descrição mais formal do Algoritmo NF^x.

Dada uma lista de caixas $L = (c_1, c_2, \ldots, c_n)$, o Algoritmo NF^x empacota as caixas de L na ordem dada. Primeiramente, a caixa c_1 é empacotada em B no canto mais próximo ao ponto $(0, 0, 0)$. A seguir, a primeira faixa é preenchida da esquerda para a direita na direção do eixo x, até que uma caixa c_i não possa ser empacotada na faixa corrente. Neste momento a caixa c_i é empacotada na segunda faixa, e este processo continua até o instante em que uma caixa c_j não possa ser empacotada na faixa corrente e nem haja espaço suficiente para criar uma nova faixa no primeiro nível. Então o Algoritmo NF^x

passa a empacotar a caixa c_j na primeira faixa do segundo nível. O processo de criar níveis e faixas continua até que todas as caixas tenham sido empacotadas.

Algoritmo $NF^x(L)$

Entrada: Constantes a e b e lista de caixas $L = (c_1, c_2, \ldots, c_n) \in \mathcal{R}(a, b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

```
% (x, y, z): Posição onde vai ser empacotada a próxima caixa.
% y_max: largura da caixa mais larga na faixa corrente.
% x_max: altura da caixa mais alta no nível corrente.
(x, y, z, y_max, z_max) ← (0, 0, 0, 0, 0);
para i ← 1 até n faça
    se (x + x_i > a) ou (y + y_i > b)
        então
            se y + y_max + y_i ≤ b
                então % gera uma nova faixa
                    y ← y + y_max;
                    (x, y_max) ← (0, 0)
                senão % gera um novo nível
                    z ← z + z_max;
                    (x, y, y_max, z_max) ← (0, 0, 0, 0);
            fim se;
    fim se;
    ℘(c_i) ← (x, y, z);
    (x, y_max, z_max) ← (x + x_i, max{y_max, y_i}, max{z_max, z_i});
fim para;
retorne ℘;
fim algoritmo.
```

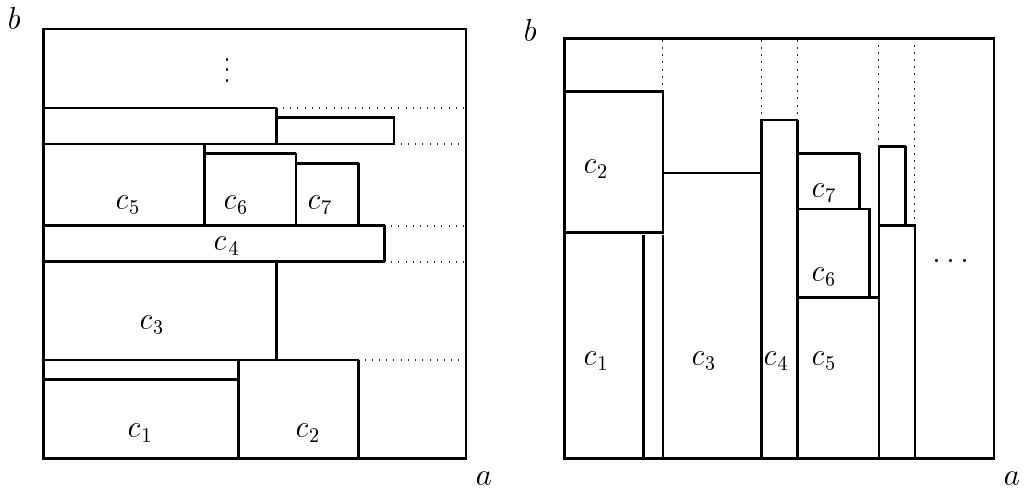
Note que todas as faixas criadas pelo Algoritmo NF^x são paralelas ao eixo x (veja Figura 3.1(a)). O Algoritmo NF^y é semelhante a NF^x, este algoritmo gera as faixas paralelas ao eixo y (veja Figura 3.1(b)).

Algoritmo $NFDH^x(L)$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}(a, b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Ordene as caixas de L em ordem não-crescente de altura.
2. $\wp \leftarrow NF^x(L)$.
3. Retorne $\wp$.

fim algoritmo.



(a) Um nível gerado por NF^x.

(b) Um nível gerado por NF^y.

Figura 3.1: Fundo do nível de um empacotamento usando a estratégia NF.

O Algoritmo $NFDH^y$ é análogo ao Algoritmo $NFDH^x$, usando NF^y ao invés da NF^x.

Apesar do Algoritmo $NFDH^f$, para o caso bidimensional, ter um limite de desempenho assintótico constante o mesmo não vale para sua versão tridimensional. De fato, como mostra o teorema a seguir, o Algoritmo $NFDH$ tem desempenho de pior caso ilimitado.

Teorema 3.1.1 *Para cada inteiro $M > 1$, existe uma instância L para o $PET(a, b)$ tal que*

$$NFDH(L) > M \cdot OPT(L).$$

Dem. Sem perda de generalidade, considere o $PET(1, 1)$. Faremos a prova para o algoritmo $NFDH^x$. Seja $L = (c_1, c_2, \ldots, c_{2k})$ tal que

$$c_i = \begin{cases} \left(1, \frac{1}{k}, 1 - (i-1)\epsilon\right) & \text{para } \; i = 1,\, 3, \ldots,\, 2k-1 \\ \\ \left(\frac{1}{k}, 1, 1 - (i-1)\epsilon\right) & \text{para } \; i = 2,\, 4, \ldots,\, 2k, \end{cases}$$

onde $k = M + 1$ e ϵ é um positivo bem pequeno.

O empacotamento gerado pelo Algoritmo $NFDH^x(L)$ dispõe cada caixa de L em um nível distinto, e portanto,

$$NFDH^x(L) = \sum_{i=1}^{2k}(1-(i-1)\epsilon) = 2k - k(2k-1)\epsilon \ .$$

Agora considere um empacotamento de dois níveis onde o primeiro nível contém todas as caixas c_i com i ímpar e o segundo nível contém todas as caixas c_i com i par. Um tal empacotamento mostra que $OPT(L) \leq 2 - \epsilon$. Logo,

$$\frac{NFDH^x(L)}{OPT(L)} \geq \frac{2k - k(2k-1)\epsilon}{2-\epsilon} \ .$$

Assim, se tomarmos $\epsilon < \frac{2}{2M^2+2M+1}$, temos o resultado desejado. No caso do Algoritmo $NFDH^y$ basta considerar a instância que resulta de L permutando-se os valores da largura e comprimento de cada caixa em L.

□

3.1.2 Algoritmo $FFDH$

No Algoritmo $NFDH$, sempre que se começa uma nova faixa, nunca mais se volta para uma faixa anteriormente criada. Um meio de se evitar possíveis "desperdícios" seria usar a mesma idéia do algoritmo de empacotamento bidimensional $FFDH^f$, no qual são sempre revisitadas as faixas anteriores. Denominaremos de $FFDH$ o algoritmo baseado nesta idéia e de $FFDH^x$ e $FFDH^y$ suas variantes. Como na seção anterior, $FFDH$ será usado para nos referirmos indistintamente a qualquer uma das variantes $FFDH^x$ e $FFDH^y$.

Vamos chamar de FF^x (respectivamente FF^y) o algoritmo correspondente à versão modificada de NF^x (respectivamente NF^y) que revisita as faixas anteriores. Mais precisamente, para decidir onde empacotar uma caixa c, este algoritmo examina as faixas começando na primeira faixa do primeiro nível. Se encontrar uma faixa onde c possa ser encaixada no fim desta, então empacota c em tal faixa. Uma nova faixa é criada (possivelmente num novo nível, se necessário) se nenhuma faixa anterior puder acomodar a caixa c.

Mais formalmente, o Algoritmo $FFDH^x$ pode ser assim descrito.

Algoritmo $FFDH^x(L)$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Ordene L em ordem não-crescente de altura.
2. $\wp \leftarrow FF^x(L)$.
3. Retorne $\wp$.

fim algoritmo.

O algoritmo $FFDH^y$ é definido analogamente, fazendo uso do algoritmo FF^y.

Mesmo sendo uma "versão melhorada" do Algoritmo $NFDH$, o Algoritmo $FFDH$ também tem um limite de desempenho de pior caso ilimitado, como mostra o próximo teorema.

Teorema 3.1.2 *Para cada inteiro $M > 1$, existe uma instância L para o $PET(a, b)$ tal que*

$$FFDH(L) > M \cdot OPT(L) \ .$$

Dem. Sem perda de generalidade, considere o $PET(1, 1)$. Faremos a prova para o algoritmo $FFDH^x$. Tome $k = 2M+1$ e considere uma lista de caixas $L = (c_1, c_2, \ldots, c_{2k})$, onde

$$c_i = \begin{cases} \left(1, \frac{1}{k} - \frac{(i-1)}{2}\delta, 1 - (i-1)\epsilon\right) & \text{para } i = 1, 3, \ldots, 2k-1 \\ \left(\frac{1}{k}, 1 - \left(\frac{1}{k} + \frac{i-2}{2}\right)\delta, 1 - (i-1)\epsilon\right) & \text{para } i = 2, 4, \ldots, 2k, \end{cases}$$

ϵ é um positivo bem pequeno e δ é tal que $0 < \delta < 1/(k(k-1))$.

Para esta instância L, o Algoritmo $FFDH^x$ gera um empacotamento de k níveis, onde cada nível i $(1 \leq i \leq k)$ contém as caixas c_{2i-1} e c_{2i}. Assim, $FFDH^x(L) = \sum_{i=1}^{k}(1 - 2(i-1)\epsilon) = k - k(k-1)\epsilon$ (veja a Figura 3.2). Por outro lado, assim como na prova do teorema anterior, temos que $OPT(L) \leq 2 - \epsilon$ (veja a Figura 3.3). Portanto,

$$\frac{FFDH^x(L)}{OPT(L)} \geq \frac{k - k(k-1)\epsilon}{2 - \epsilon} \ .$$

Basta então tomarmos $\epsilon < \frac{1}{4M^2+M}$, para garantirmos que $FFDH^x(L) > M \cdot OPT(L)$. No caso do algoritmo $FFDH^y$ a prova é análoga.

□

Muitos dos algoritmos de empacotamento bidimensional geram resultados melhores quando a instância contém apenas quadrados. Assim, se tivermos que $L \in \mathcal{Q}_m(a, b)$, i.e., a instância consiste de caixas com fundo mais regulares, é de se esperar que os algoritmos de empacotamento tridimensional dêem resultados melhores. Infelizmente isto não acontece, como mostra o teorema abaixo.

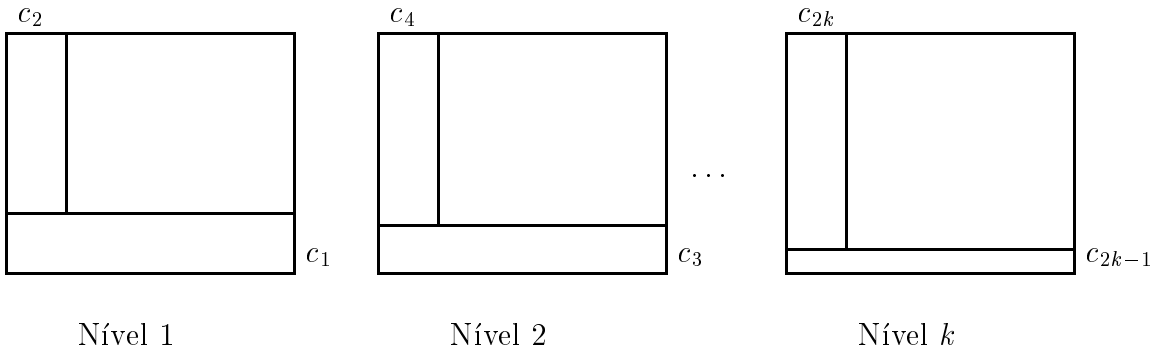


Figura 3.2: Empacotamento da lista L gerado pelo Algoritmo $FFDH^x$.

Figura 3.3: Um empacotamento ótimo da lista L.

Teorema 3.1.3 *Para cada inteiro $M > 1$, existe uma instância $L \in \mathcal{Q}_m(a,b)$ para o $PET(a,b)$, $m \geq 1$, tal que $NFDH(L) > M \cdot OPT(L)$ e $FFDH(L) > M \cdot OPT(L)$.*

Dem. Sem perda de generalidade, considere o $PET(1,1)$. Seja $k > \max\{2M+1, m\}$, $n = k^2(1+(k-1)^2)$, $L = \{c_1, c_2, \ldots, c_n\}$, onde $c_i = (x_i, y_i, z_i)$ é tal que

$$(x_i, y_i) = \begin{cases} \left(\frac{1}{k}, \frac{1}{k}\right) & \text{se } \ i \bmod (1+(k-1)^2) = 1 \ , \\ \left(\frac{1}{k(k-1)}, \frac{1}{k(k-1)}\right) & \text{caso contrário .} \end{cases}$$

Faça com que $z_1 > z_2 > \cdots > z_n$ e, em particular,

$$z_{(i-1)n_1+1} = 1 - i\delta \ \text{ para } \ 1 \leq i \leq n,$$

onde

$$n_1 = k(1+(k-1)^2) \quad \text{e} \quad \delta = \frac{2}{k(k+1)} \ .$$

Para esta instância L, ambos os algoritmos $NFDH$ e $FFDH$ produzem um empacotamento com k níveis. Não é difícil verificar que

$$NFDH(L) = FFDH(L) = k - \frac{k(k+1)\delta}{2} = k - 1 \ .$$

Considere um empacotamento de L em dois níveis, o primeiro com todas as caixas de fundo $\left(\frac{1}{k}, \frac{1}{k}\right)$ e o segundo com todas as caixas de fundo $\left(\frac{1}{k(k-1)}, \frac{1}{k(k-1)}\right)$. Claramente a altura deste empacotamento é menor que $2(1-\delta)$ Assim,

$$OPT(L) < 2 - 2\delta \ .$$

Pela escolha de k e δ segue que

$$\frac{NFDH(L)}{OPT(L)} = \frac{FFDH(L)}{OPT(L)} > \frac{k-1}{2-2\delta} > \frac{k-1}{2} > M. \qquad \square$$

3.2 Usando garantia de área mínima por nível

O principal fator que faz com que os algoritmos $NFDH$ e $FFDH$ tenham limites de desempenho de pior caso ilimitado é o fato desses algoritmos não garantirem um mínimo de utilização (de volume) em cada nível. Os próximos algoritmos que analisaremos nesta seção empacotam as caixas garantindo uma percentagem mínima de utilização (de volume) em cada nível. Os lemas abaixo nos fornecem uma relação importante entre a área de fundo garantida em um nível com o limite de desempenho assintótico de um algoritmo para o $PET(a,b)$. Estes resultados encontram-se em [23].

Lema 3.2.1 *Para qualquer instância L do $PET(a,b)$ tem-se que*

$$OPT(L) \geq \frac{V(L)}{ab}.$$

□

Lema 3.2.2 *Seja L uma instância para o $PET(a,b)$ e seja $\wp$ um empacotamento de L. Suponha que $\wp$ é dividido em níveis $N_1, N_2, \ldots, N_v$, onde cada nível N_i representa um empacotamento $\wp_i$ de uma lista L_i, tal que $\wp = \wp_1 \| \wp_2 \| \cdots \| \wp_v$, $L = L_1 \cup L_2 \cup \cdots \cup L_v$ e $\min\{z(c) : c \in L_i\} \geq \max\{z(c), c \in L_{i+1}\}$, $1 \leq i \leq v-1$. Ademais, suponha que existe uma constante positiva A_{lbound} tal que $S(L_i) \geq A_{lbound}$, $1 \leq i \leq v-1$. Então*

$$H(\wp) \leq \frac{V(L)}{A_{lbound}} + Z,$$

onde $Z = \max\{z(c) : c \in L\}$.

Dem. Por hipótese, cada lista L_i é empacotada em um nível. Como $S(L_i) \geq A_{lbound}$ todas as caixas de L_i têm altura maior ou igual à das caixas de L_{i+1}, $1 \leq i \leq v-1$, temos que

$$V(L_i) \geq A_{lbound} \cdot H(\wp_{i+1}) \ , \quad \text{para} \quad 1 \leq i \leq v-1 \ .$$

Somando essas desigualdades obtemos a seguinte desigualdade para o volume.

$$\begin{aligned} V(L) &= V(L_1) + V(L_2) + \cdots + V(L_v) \\ &> A_{lbound} \left(\sum_{i=2}^{v} H(\wp_i) \right) \\ &= A_{lbound} \left(\sum_{i=1}^{v} H(\wp_i) - H(\wp_1) \right). \end{aligned}$$

Portanto, $H(\wp) = \sum_{i=1}^{v} H(\wp_i) < \frac{V(L)}{A_{lbound}} + H(\wp_1) = \frac{V(L)}{A_{lbound}} + Z$, o que completa a demonstração do lema.

□

A partir dos Lemas 3.2.1 e 3.2.2 podemos achar uma relação entre a garantia de área por nível (nas condições do Lema 3.2.2) e a altura de um empacotamento ótimo.

Corolário 3.2.3 *Seja L uma instância para o $PET(a,b)$ e seja $\wp$ um empacotamento nas condições do Lema 3.2.2. Então*

$$H(\wp) \leq \frac{ab}{A_{lbound}} OPT(L) + Z,$$

onde $Z = \max\{z(c) : c \in L\}$.

Dem. Segue diretamente da aplicação do Lema 3.2.1 e do Lema 3.2.2. □

3.2.1 Algoritmo G_1

Alguns teoremas do Capítulo 2 fornecem condições suficientes para garantir o empacotamento de uma lista de retângulos, *sob certas condições de área*, pelo Algoritmo $NFDH^p$ (o mesmo vale para o Algoritmo $NFDW^p$).

Baseando-se nestes teoremas, Li e Cheng [22] construíram os algoritmos G^x e G^y para o $PET(a,b)$, descritos a seguir. Ambos os algoritmos têm como parâmetros as constantes A_{lbound} e A_{ubound}. A constante A_{ubound} é a área de fundo máxima de um conjunto de caixas para que os algoritmos $NFDH^p$ e $NFDW^p$ consigam empacotar em um nível, respeitadas as respectivas condições; e A_{lbound} é a área mínima que G^x (G^y) vai garantir em cada nível.

Algoritmo $G^x(L_x, A_{lbound}, A_{ubound})$

Entrada: Constantes a, b, A_{lbound} e A_{ubound} e lista de caixas $L_x \in \mathcal{R}(a,b)$, .
Saída: Empacotamento $\wp_x$ de L_x em $B = (a, b, \infty)$.

1. Divida L_x em dois subconjuntos L_{x1} e L_{x2}, sendo

$$\begin{aligned} L_{x1} &= \{c_i \ : x_i y_i \leq A_{lbound}\} \text{ e} \\ L_{x2} &= \{c_i \ : x_i y_i > A_{lbound}\} \ . \end{aligned}$$

Sem perda de generalidade, considere $L_{x1} = \{c_1, \ldots, c_p\}$ e $L_{x2} = \{c_{p+1}, \ldots, c_n\}$.
2. Ordene L_{x1} em ordem não-crescente de altura. Assuma que $L_{x1} = (c_1, \ldots, c_p)$.
3. Divida a lista L_{x1} em sublistas $L_1, \ldots, L_v$, onde

$$\begin{aligned} & L_i = (c_{k_{i-1}+1}, c_{k_{i-1}+2}, \ldots, c_{k_i}) \ , \quad i \leq i \leq v \ , \\ & k_0 = 0 \ , \quad k_v = p \ \text{ e } \quad \text{para } \ 1 \leq i \leq v-1 \ \text{ cada } \ k_i \ \text{ é tal que} \\ & \sum_{j=k_{i-1}+1}^{k_i} x_j y_j \leq A_{ubound} \quad \text{e} \quad \sum_{j=k_{i-1}+1}^{k_i+1} x_j y_j > A_{ubound} \ . \end{aligned}$$

4. Ordene cada L_i em ordem não-crescente de largura.
5. Empacote cada L_i, $(1 \leq i \leq v)$ em um nível, usando o Algoritmo $NFDH^p$ aplicado à lista dos fundos das caixas em L_i. Seja $\wp_{x1}$ o empacotamento assim gerado.
6. Empacote cada caixa de L_{x2} em um nível distinto, gerando um empacotamento $\wp_{x2}$.
7. $\wp_x \leftarrow \wp_{x1} || \wp_{x2}$.
8. Retorne $\wp_x$.

fim algoritmo.

O Algoritmo G^y é definido analogamente ao algoritmo G^x, usando o algoritmo $NFDW^p$.

A partir dos algoritmos G^x e G^y podemos construir um algoritmo mais genérico, chamado G, descrito a seguir .

Algoritmo $G(L, A_{lbound}, A_{ubound})$

Entrada: Constantes a, b, A_{lbound} e A_{ubound} e lista de caixas $L \in \mathcal{R}(a, b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Divida L em dois subconjuntos L_x e L_y, tomando

$$\begin{aligned} L_x &= \{c_i \ : x_i \geq y_i\} \ \text{ e} \\ L_y &= \{c_i \ : x_i < y_i\} \ . \end{aligned}$$

2. $\wp_x \leftarrow G^x(L_x, A_{lbound}, A_{ubound})$.
3. $\wp_y \leftarrow G^y(L_y, A_{lbound}, A_{ubound})$.
4. $\wp \leftarrow \wp_x \| \wp_y$.
5. Retorne $\wp$.

fim algoritmo.

No Capítulo 2 vimos alguns resultados sobre algoritmos de empacotamento bidimensional, que garantem que se a área de uma lista de retângulos S não ultrapassa certo valor, então os retângulos em S podem ser empacotados em apenas um nível. Mais precisamente, o Corolário 2.2.13 garante que se uma lista de retângulos $L = (c_1, \ldots, c_m)$ tem área total de no máximo $\frac{7}{16}$, e $x_i \geq y_i$, então o Algoritmo $NFDH^p$ consegue empacotar a lista L em um quadrado unitário. Baseados neste resultado, que pode ser combinado com o Corolário 3.2.3, Li e Cheng [22] desenvolveram o Algoritmo G_1, descrito a seguir.

Algoritmo $G_1(L)$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}(a, b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Reparametrize o fundo da caixa (a, b, ∞) para uma caixa com fundo $(1, 1)$; reparametrize o fundo das caixas da lista L na mesma proporção, obtendo uma lista L'.
2. $\wp' \leftarrow G\left(L', \frac{7}{32}, \frac{7}{16}\right)$.
3. Retorne à parametrização original, obtendo $\wp$ a partir de $\wp'$.
4. Retorne $\wp$.

fim algoritmo.

A reparametrização foi necessária, pois o Algoritmo $NFDH^p$ garante $\frac{7}{32}$ quando $(a, b) = (1, 1)$. Como devemos aplicar G^x e G^y, i.e., empacotar na direção x e depois

na direção y, a melhor forma de obter garantias de áreas iguais é trabalhar em um problema onde $a = b$, i.e., reparametrizar.

Teorema 3.2.4 *Para qualquer instância L do $PET(a, b)$, tal que nenhuma caixa tem altura maior que Z, tem-se que*

$$G_1(L) \leq \frac{32}{7} OPT(L) + 2Z \ .$$

Dem. A reparametrização (do fundo) das caixas da instância L de $PET(a, b)$ para uma instância L' de $PET(1, 1)$ não muda em nada a altura do empacotamento.

O Algoritmo G (chamado pelo Algoritmo G_1) divide L em dois subconjuntos L_x e L_y, e então aplica G^x em L_x e G^y em L_y. No empacotamento de L_x temos por sua vez que G^x divide L_x em L_{x1} e L_{x2}, gerando os empacotamentos $\wp_{x1}$ e $\wp_{x2}$, respectivamente.

A área de fundo de cada nível de $\wp_{x2}$ é pelo menos $\frac{7}{32}$, e portanto, $V(L_{x2}) \geq \frac{7}{32} H(\wp_{x2})$. Por outro lado, temos pelo Lema 3.2.2 que $H(\wp_{x1}) \leq \frac{32}{7} V(L_{x1}) + Z$. Assim,

$$\begin{aligned} H(\wp_x) &= H(\wp_{x1}) + H(\wp_{x2}) \\ &\leq \frac{32}{7}(V(L_{x1}) + V(L_{x2})) + Z \\ &\leq \frac{32}{7} V(L_x) + Z \ . \end{aligned}$$

Da mesma forma, para L_y, temos que

$$H(\wp_y) \leq \frac{32}{7} V(L_y) + Z \ .$$

Portanto,

$$\begin{aligned} H(\wp) &= H(\wp_x) + H(\wp_y) \\ &\leq \frac{32}{7} V(L) + 2Z \ , \end{aligned}$$

e pelo Lema 3.2.1, concluimos que

$$H(\wp) \leq \frac{32}{7} OPT(L) + 2Z \ .$$

□

O Teorema 3.2.4 nos fornece uma cota superior para o limite de desempenho assintótico do Algoritmo G_1. Li e Cheng [22] não provaram que este valor é justo, mas obtiveram o seguinte resultado.

Corolário 3.2.5 *O limite de desempenho assintótico do Algoritmo G_1, $r(G_1)$, é tal que* $r(G_1) \in \left[\frac{13}{3}, \frac{32}{7}\right]$.

Dem. Sem perda de generalidade, considere o $PET(1,1)$. É suficiente exibir uma familia de instâncias L_δ tal que $G_1(L_\delta) = \frac{13}{3}OPT(L_\delta)$. Seja L_δ um conjunto de caixas $L_\delta = L_\delta^1 \cup L_\delta^2$, onde L_δ^1 contém $4n$ caixas de tamanho $\left(1, \frac{7}{32} + \delta, 1\right)$ e L_δ^2 contém n caixas de tamanho $\left(1, \frac{1}{8} - 4\delta, 1\right)$, sendo $\delta < \frac{1}{256}$ e $n \bmod 3 = 0$.

No empacotamento gerado por G cada caixa de L_δ^1 ocupa um nível distinto e cada três caixas de L_δ^2 ocupam um nível distinto, sendo tais níveis todos distintos. Assim, temos que

$$G_1(L_\delta) = 4n + \frac{n}{3} = \frac{13}{3}n \ .$$

O empacotamento ótimo tem n níveis, cada um contendo quatro caixas de L_δ^1 e uma caixa de L_δ^2. Logo, $OPT(L_\delta) = n$, e portanto,

$$G_1(L_\delta) = \frac{13}{3}OPT(L_\delta) \ .$$

□

3.2.2 Algoritmo $G_m,\ m \geq 3$

Um aspecto importante é o comportamento do Algoritmo G^x quando as caixas a serem empacotadas obedecem certas restrições quanto ao fundo. Consideraremos aqui o caso em que as caixas c_i são tais que

$$x_i \leq \frac{a}{m} \ , \quad y_i \leq \frac{b}{m} \ , \quad i = 1, 2, \ldots, n \ .$$

Veremos que neste caso o algoritmo G_m – descrito a seguir – que faz uso do algoritmo G^x, tem um limite de desempenho tanto melhor quanto maior o valor de m (veja [22]).

Algoritmo $G_m(L)$, $m \geq 3$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}_m(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. $\wp \leftarrow G^x\left(L, \left[\left(1 - \frac{1}{m}\right)^2 - \frac{1}{m^2}\right] ab, \left(1 - \frac{1}{m}\right)^2 ab\right)$.
2. Retorne $\wp$.

fim algoritmo.

Lema 3.2.6 *Se $L \in \mathcal{R}_m(a,b)$, $m \geq 3$, e nenhuma caixa de L tem altura maior que Z, então*

$$G_m(L) < \left(\frac{m}{m-2}\right)\frac{V(L)}{ab} + Z \ .$$

Dem. Pelo Teorema 2.2.14, se uma lista de caixas L' tiver área acumulativa menor ou igual a $\left(1-\frac{1}{m}\right)^2 ab$, e cada caixa c_i em L' é tal que $x_i \leq \frac{a}{m}$, $y_i \leq \frac{b}{m}$, então o Algoritmo $NFDH^p$ empacota L' em um único nível. Note também que não existem caixas em L com área superior a $\left[\left(1-\frac{1}{m}\right)^2 - \frac{1}{m^2}\right] ab$. Sejam $L_1, \ldots, L_v$ as sublistas de L geradas pelo Algoritmo G^x, onde $L_i = (c_{k_i}, c_{k_i+1}, \ldots, c_{k_{i+1}-1})$. Como $S(L_i) + S(c_{k_{i+1}}) > \left(1-\frac{1}{m}\right)^2 ab$ e $S(c_{k_{i+1}}) \leq \frac{ab}{m^2}$, temos que

$$S(L_i) > \left(1-\frac{1}{m}\right)^2 ab - \frac{ab}{m^2} \ .$$

Além disso, este empacotamento satisfaz as condições do Lema 3.2.2; portanto,

$$G_m(L) < \frac{V(L)}{\left[\left(1-\frac{1}{m}\right)^2 - \frac{1}{m^2}\right] ab} + Z = \left(\frac{m}{m-2}\right)\frac{V(L)}{ab} + Z \ .$$

□

Teorema 3.2.7 *Para todo $L \in \mathcal{R}_m(a,b)$, $m \geq 3$, tal que nenhuma caixa tem altura maior que Z, tem-se*

$$G_m(L) < \left(\frac{m}{m-2}\right) OPT(L) + Z \ .$$

Mais ainda, este limite é justo.

Dem. A desigualdade segue do Lema 3.2.1 e do Lema 3.2.6. Para mostrar a justeza do limite, sem perda de generalidade, considere o $PET(1,1)$. Para cada $m \geq 3$ considere a instância $L = L(m) = (c_1, \ldots, c_n)$, onde $n = k^2m^2(m-1)^2$, $k > 0$,

$$c_i = \begin{cases} \left(\frac{1}{k}, \frac{1}{k}, 1-(i-1)\epsilon\right) & \text{se } i \bmod (m-1)^2 = 0; \\ \left(\frac{1}{m}, \frac{1}{m}, 1-(i-1)\epsilon\right) & \text{caso contrário;} \end{cases}$$

sendo ϵ um número bem pequeno.

Vamos chamar de X as caixas com fundo $\left(\frac{1}{m}, \frac{1}{m}\right)$ e de Y as caixas com fundo $\left(\frac{1}{k}, \frac{1}{k}\right)$, onde $k \to \infty$.

Esta instância foi construída de modo a forçar que o Algoritmo G_m garanta pouca área por nível. Assim, consideramos caixas grandes X e caixas pequenas Y, de modo que o algoritmo deixa de empacotar uma caixa X por muito pouco.

O Passo 3 do Algoritmo G^x divide a instância L em $v = k^2m^2$ grupos, onde cada grupo contém $(m-1)^2 - 1$ caixas X e uma caixa Y, conforme ilustrado abaixo.

$$L = \underbrace{\left(\underbrace{X, X, \ldots, X, Y}_{g=(m-1)^2}, \underbrace{X, X, \ldots, X, Y}_{g=(m-1)^2}, \ldots, \underbrace{X, X, \ldots, X, Y}_{g=(m-1)^2} \right)}_{v=k^2m^2 \text{ grupos}}$$

Cada lista L_i, $(1 \leq i \leq v)$ produzida no Passo 3 é tal que $S(L_i) = S((X, X, \ldots, X, Y))$ $\leq \left(1 - \frac{1}{m}\right)^2 ab$ e $S(L_i) + S(X) > \left(1 - \frac{1}{m}\right)^2 ab$.

Assim, temos v níveis onde cada nível i tem altura $z_{(i-1)g+1} = 1 - (i-1)g\epsilon$, sendo $g = (m-1)^2$. Portanto,

$$G_m(L) = z_1 + z_{g+1} + z_{2g+1} + \cdots + z_{(v-1)g+1} = v - \frac{v(v-1)}{2} g\epsilon \ . \tag{3.1}$$

Considere outro empacotamento com $\frac{v[m-1)^2-1]}{m^2} = k^2[(m-1)^2 - 1]$ níveis, cada qual contendo m^2 caixas X, e $\frac{v}{k^2} = m^2$ níveis, cada qual contendo k^2 caixas Y.

Como a altura de cada nível é menor que 1, segue que

$$OPT(L) < k^2[(m-1)^2 - 1] + m^2 \ . \tag{3.2}$$

De (3.1) e de (3.2) temos que

$$\lim_{\epsilon \to 0} \frac{G_m(L)}{OPT(L)} = \frac{k^2m^2}{k^2[(m-1)^2 - 1] + m^2} \ .$$

É fácil ver que este valor é menor que $\frac{m}{m-2}$ e pode se tornar bem próximo de $\left(\frac{m}{m-2}\right)$ à medida que $k \to \infty$.

□

3.2.3 Algoritmo $GQ_m,\ m \geq 1$

Quando as caixas têm fundo quadrado, podemos usar o Teorema 2.2.9 e o Teorema 2.2.11 para garantir área mínima em cada nível. Esta é a idéia que está por trás do Algoritmo

GQ_m, descrito a seguir, para empacotar listas L tal que $L \in \mathcal{Q}_m$.

Algoritmo $GQ_m(L),\ m \geq 1$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{Q}_m(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Se $m = 1$ então
$\wp \leftarrow G^x\left(L, \frac{ab}{4}, \frac{ab}{2}\right)$.
2. Se $m \geq 2$ então
$\wp \leftarrow G^x\left(L, \left(\frac{m-1}{m}\right)^2 ab, \left[\frac{1}{m^2} + \left(\frac{m-1}{m}\right)^2\right] ab\right)$.
3. Retorne $\wp$.

fim algoritmo.

Veremos a seguir um resultado devido a Li e Cheng [22], sobre o limite de desempenho assintótico do algoritmo GQ_m.

Lema 3.2.8 *Seja $L \in \mathcal{Q}_m(a,b)$, $m \geq 1$, uma lista tal que nenhuma caixa tem altura maior que Z. Então*

$$GQ_m(L) \leq \alpha_m \frac{V(L)}{ab} + Z\ ,$$

onde

$$\alpha_m = \begin{cases} 4 & \text{se } \ m = 1, \\ \left(\frac{m}{m-1}\right)^2 & \text{se } \ m \geq 2. \end{cases}$$

Dem. A demonstração deste resultado é análoga à demonstração do Teorema 3.2.4. □

Teorema 3.2.9 *Para toda lista $L \in \mathcal{Q}_m(a,b)$, $m \geq 1$, tal que nenhuma caixa tem altura maior que Z, temos que*

$$GQ_m(L) \ \leq \ \alpha_m OPT(L) + Z,$$

onde

$$\alpha_m = \begin{cases} 4 & \text{se } \ m = 1, \\ \left(\frac{m}{m-1}\right)^2 & \text{se } \ m \geq 2. \end{cases}$$

Mais ainda, este limite é justo.

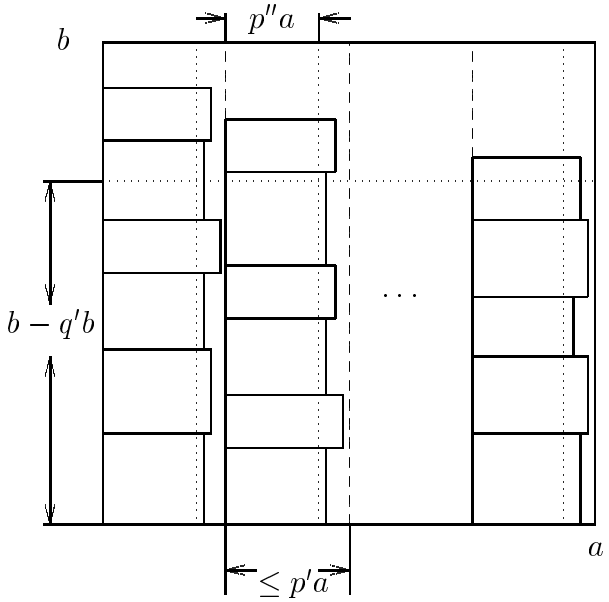


Figura 3.4: Um nível em um empacotamento $NFDH^y$ para $L \subseteq \mathcal{C}\,[p'', p'\,;\, 0, q'](a, b)$.

Dem. A desigualdade segue aplicando-se o Lema 3.2.1 e o Lema 3.2.8. Para provar a justeza do limite, para cada $m \geq 2$ considere a instância $L_m = (c_1, \ldots, c_n)$, onde $n = k^2m^2[1 + (m-1)^2]$, $k \geq m$, e

$$c_i = \begin{cases} \left(\frac{1}{k}, \frac{1}{k}, 1 - (i-1)\epsilon\right) & \text{se } i \bmod 1 + (m-1)^2 = 0, \\ \left(\frac{1}{m}, \frac{1}{m}, 1 - (i-1)\epsilon\right) & \text{caso contrário.} \end{cases}$$

Note que $\alpha_1 = \alpha_2$. Assim, podemos considerar a instância L_2 para provar que os limites α_1 e α_2 são justos.

Deixaremos a prova a cargo do leitor, já que esta pode ser feita de maneira análoga à prova do Teorema 3.2.7. □

3.2.4 Algoritmo C

Descreveremos a seguir um algoritmo que chamaremos de Algoritmo C, desenvolvido por Li e Cheng [22], que divide a lista L em várias sublistas, e aplica em cada uma destas sublistas um algoritmo apropriado. Dois destes algoritmos são os algoritmos $NFDH^x$ e $NFDH^y$.

Já vimos que o Algoritmo $NFDH^x$ tem limite de desempenho de pior caso ilimitado. Entretanto, para instâncias onde as caixas obedecem certas restrições, descritas a seguir,

o Algoritmo $NFDH^x$ deixa de ter um tal desempenho. Isto também é válido para o Algoritmo $NFDH^y$.

Nos próximos dois lemas veremos que o algoritmo $NFDH$ nos dá um limite superior para a altura do empacotamento quando a instância está contida no conjunto $\mathcal{C}\,[p'', p'\ ;\ q'', q'](a, b)$, onde $\frac{1}{p'}$ e $\frac{1}{q'}$ são inteiros. Lembramos aqui que

$$\mathcal{C}\,[p'', p'\ ;\ q'', q'](a, b) = \{c_i = (x_i, y_i, z_i) :\ p''a < x_i \leq p'a\ ,\ \ q''b < y_i \leq q'b\}\ ,$$

onde $0 \leq p'' < p' \leq 1,\ \ 0 \leq q'' < q' \leq 1,\ \ a > 0\ \ \text{e}\ \ b > 0$.

Lema 3.2.10 *Se $\wp$ é um empacotamento gerado pelo Algoritmo $NFDH^d$, $d \in \{x, y\}$, aplicado a uma instância $L \subseteq \mathcal{C}\,[p'', p'\ ;\ q'', q'](a, b)$, onde $\frac{1}{p'}$ e $\frac{1}{q'}$ são inteiros e $Z = \max\{z(c) :\ c \in L\}$, então*

$$H(\wp) \leq \frac{p'}{p''}\frac{q'}{q''}\frac{V(L)}{ab} + Z\ .$$

Dem. Como $q''b < y_i \leq q'b$ e $p''b < x_i \leq p'b$, para cada caixa c_i em L, cada nível de $\wp$ contém $\frac{1}{q'}$ faixas e cada faixa contém $\frac{1}{p'}$ caixas. Portanto, por nível (exceto talvez o último nível) temos $\frac{1}{q'}\frac{1}{p'}$ caixas, cada caixa com área de fundo superior à $q''p''ab$. Assim, a área utilizada em cada nível é de pelo menos $\frac{q''p''ab}{q'p'}$. A demonstração segue aplicando-se o Lema 3.2.2.

□

Doravante assumiremos que se $d \in \{x, y\}$ então $\overline{d} = \{x, y\} \setminus d$.

Lema 3.2.11 *Para $d \in \{x, y\}$, seja $\wp^d$ o empacotamento gerado pelo Algoritmo $NFDH^{\overline{d}}$ aplicado a uma instância L^d, onde nenhuma caixa tem altura maior que Z.*

(a) *Se $L^y \subseteq \mathcal{C}\,[p'', p'\ ;\ 0, q'](a, b)$ e $q' < 1$ então $H(\wp^y) \leq \frac{p'}{p''(1-q')}\frac{V(L^y)}{ab} + Z$.*

(b) *Se $L^x \subseteq \mathcal{C}\,[0, p'\ ;\ q'', q'](a, b)$ e $p' < 1$ então $H(\wp^x) \leq \frac{q'}{q''(1-p')}\frac{V(L^x)}{ab} + Z$.*

Em ambos os casos estamos supondo que $\frac{1}{p'}$ e $\frac{1}{q'}$ são inteiros.

Dem. Consideremos o Caso (a). Seja $L^y = (c_1, \ldots, c_n)$. Como $p''a < x_i \leq p'a$, há pelo menos $\frac{1}{p'}$ faixas paralelas ao eixo y. Como $y_i \leq q'b$, temos que a soma das larguras das caixas em uma faixa é de pelo menos $b - q'b$ (veja a Figura 3.4). Logo, a área total ocupada em cada nível, exceto talvez o último nível, é de pelo menos $\frac{1}{p'}p''a(b - q'b)$. A demonstração segue aplicando-se o Lema 3.2.2.

A demonstração para o Caso (b) é análoga.

□

Corolário 3.2.12 *Se $\wp$ é um empacotamento gerado pelo Algoritmo $NFDH^y$ aplicado a uma instância $L \in \mathcal{C}\left[\frac{1}{m+1}, \frac{1}{m}\ ;\ 0, \frac{1}{m}\right](a,b)$, $m \geq 2$, e $Z = \max\{z(c) :\ c \in L\}$ então*

$$H(\wp) \leq \left(\frac{m+1}{m-1}\right)\frac{V(L)}{ab} + Z\ .$$

O mesmo resultado vale para um empacotamento gerado pelo Algoritmo $NFDH^x$ aplicado a uma instância $L \subseteq \mathcal{C}\left[0, \frac{1}{m}\ ;\ \frac{1}{m+1}, \frac{1}{m}\right](a,b)$.

□

Veremos agora um outro algoritmo que é usado pelo Algoritmo C. Trata-se de um algoritmo extremamente simples, denominado de **UC (Uma Coluna)**.

Algoritmo $UC(L)$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Construa um empacotamento $\wp$ colocando as caixas de L uma sobre a outra.
2. Retorne $\wp$.

fim algoritmo.

Os seguintes resultados relacionados a este algoritmo são triviais, mas serão mencionados para uso posterior.

Lema 3.2.13 *Se s é uma constante e $S(c) \geq s$, para toda caixa c em $L = (c_1, \ldots, c_n)$, então*

$$UC(L) \leq \frac{V(L)}{s}\ .$$

Dem. $V(L) = \sum_{c\in L} x(c)y(c)z(c) = \sum_{c\in L} S(c)z(c) \leq s\sum_{c\in L} z(c) = s \cdot UC(L)$.

□

Lema 3.2.14 *Seja L uma lista e L_1 uma sublista de L, $L_1 = (c_1, c_2, \ldots, c_n)$, onde $x_i > \frac{a}{2}$ e $y_i > \frac{b}{2}$ $(i = 1, \ldots, n)$. Então se $\wp_1$ é o empacotamento gerado pelo Algoritmo UC aplicado à lista L_1, temos que*

$$OPT(L) \geq H(\wp_1)\ .$$

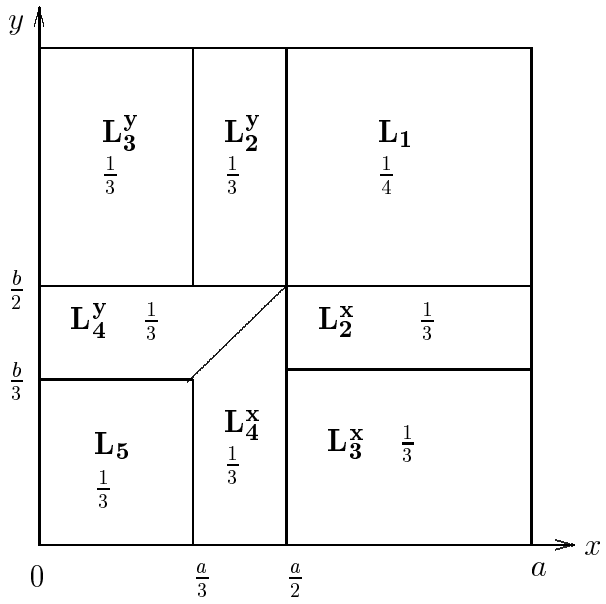


Figura 3.5: Partição da lista L efetuada pelo Algoritmo C.

Dem. Isto é claro, já que duas caixas de L_1 não podem ser empacotadas lado a lado.

□

O Algoritmo C, descrito a seguir, é baseado na observação de que quando as caixas têm fundos similares o Algoritmo $NFDH$ e suas variantes podem dar bons empacotamentos. Assim, este algoritmo divide a instância L em oito sublistas e aplica um algoritmo apropriado para cada sublista.

Na Figura 3.5 é indicada a partição de L efetuada pelo Algoritmo C. Nesta figura, a fração $\frac{p}{q}$ indicada em cada região significa que o algoritmo gera um empacotamento da sublista correspondente com área de fundo de pelo menos $\frac{p}{q}ab$. Isto ficará claro na análise do desempenho do algoritmo. Esta mesma explicação vale para as figuras de outros algoritmos que serão apresentados mais tarde.

Algoritmo $C(L)$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Particione a lista L em oito sublistas como segue (veja a Figura 3.5).

$$L_1 = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ \tfrac{1}{2},1\right](a,b), \qquad L_5 = L \bigcap \mathcal{C}\left[0,\tfrac{1}{3}\ ;\ 0,\tfrac{1}{3}\right](a,b),$$
$$L_2^x = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ \tfrac{1}{3},\tfrac{1}{2}\right](a,b), \qquad L_2^y = L \bigcap \mathcal{C}\left[\tfrac{1}{3},\tfrac{1}{2}\ ;\ \tfrac{1}{2},1\right](a,b)$$
$$L_3^x = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ 0,\tfrac{1}{3}\right](a,b), \qquad L_3^y = L \bigcap \mathcal{C}\left[0,\tfrac{1}{3}\ ;\ \tfrac{1}{2},1\right](a,b),$$
$$L_4^x = L \bigcap \mathcal{C}\left[\tfrac{1}{3},\tfrac{1}{2}\ ;\ 0,\tfrac{1}{2}\right](a,b) \bigcap \mathcal{X}(a,b), \quad L_4^y = L \bigcap \mathcal{C}\left[0,\tfrac{1}{2}\ ;\ \tfrac{1}{3},\tfrac{1}{2}\right](a,b) \bigcap \mathcal{Y}(a,b).$$

2. $\wp_1 \leftarrow UC(L_1)$.
3. $\wp_i^x \leftarrow NFDH^y(L_i^x)$, para $i = 2,3,4$.
4. $\wp_i^y \leftarrow NFDH^x(L_i^y)$, para $i = 2,3,4$.
5. $\wp_5 \leftarrow G_3(L_5)$.
6. $\wp = \wp_1 \| \wp_2^x \| \wp_2^y \| \cdots \| \wp_4^x \| \wp_4^y \| \wp_5$.
7. Retorne $\wp$.

fim algoritmo.

Como já mencionamos, o algoritmo C foi desenvolvido por Li e Cheng [22] que também provaram o seguinte resultado.

Teorema 3.2.15 *Para qualquer instância L do $PET(a,b)$, na qual nenhuma caixa tem altura maior que Z, temos que*

$$C(L) \leq 3,25 \cdot OPT(L) + 7Z \ .$$

Mais ainda, este limite é justo.

Dem. No Passo 2 do algoritmo C temos que $\wp_1$ é o empacotamento de L_1, gerado pelo Algoritmo UC. Como cada caixa de L_1 tem pelo menos $\frac{ab}{4}$ de área, pelo Lema 3.2.13 segue que

$$V(L_1) \geq \frac{ab}{4} H(\wp_1) \ .$$

Seja $h_1 = H(\wp_1)$. Reescrevendo a desigualdade acima, temos que

$$\frac{h_1}{4} \leq \frac{V(L_1)}{ab} \ . \tag{3.3}$$

No Passo 3 é gerado o empacotamento $\wp_i^d$ da lista L_i^d aplicando o Algoritmo $NFDH^{\overline{d}}$, $i = 2,3$, $d = x,y$. Assim, aplicando o Lema 3.2.10 às listas L_2^x e L_2^y, temos que

$$H(\wp_2^x) \ \leq \ 3\frac{V(L_2^x)}{ab} + Z \ , \tag{3.4}$$

$$\tag{3.5}$$

$$H(\wp_2^y) \ \leq \ 3\frac{V(L_2^y)}{ab} + Z \ . \tag{3.6}$$

Aplicando o Lema 3.2.11 às listas L_3^x, L_3^y, $\wp_4^x$ e $\wp_4^y$ temos

$$H(\wp_3^x) \leq 3\frac{V(L_3^x)}{ab} + Z \ , \tag{3.7}$$

$$H(\wp_3^y) \leq 3\frac{V(L_3^y)}{ab} + Z \ , \tag{3.8}$$

$$H(\wp_4^x) \leq 3\frac{V(L_4^x)}{ab} + Z \ , \tag{3.9}$$

$$H(\wp_4^y) \leq 3\frac{V(L_4^y)}{ab} + Z \ . \tag{3.10}$$

No Passo 5 é aplicado o Algoritmo G_3 em L_5. Como $L_5 \in \mathcal{R}_3(a,b)$, segue pelo Lema 3.2.6 que

$$H(\wp_5) \leq 3\frac{V(L_5)}{ab} + Z \ . \tag{3.11}$$

Fazendo $H = \sum_{i=2,\ldots,4} \sum_{d=x,y} H(\wp_i^d) \ + \ H(\wp_5) \ - \ 7Z$ e usando as desigualdades (3.4) à (3.11), temos que

$$\frac{1}{3}H \leq \frac{1}{ab} \sum_{i=2,\ldots,4} \ \sum_{d=x,y} V(L_i^d) \ + \ V(L_5) \ . \tag{3.12}$$

Usando as desigualdades (3.3) e (3.12), concluimos que

$$\begin{aligned} \frac{1}{4}h_1 + \frac{1}{3}H &\leq \frac{1}{ab}V(L_1) + \frac{1}{ab}V(L \setminus L_1) \\ &\leq \frac{V(L)}{ab} \\ &\leq OPT(L) \ , \end{aligned}$$

i.e.,

$$OPT(L) \geq \frac{1}{4}h_1 + \frac{1}{3}H \ . \tag{3.13}$$

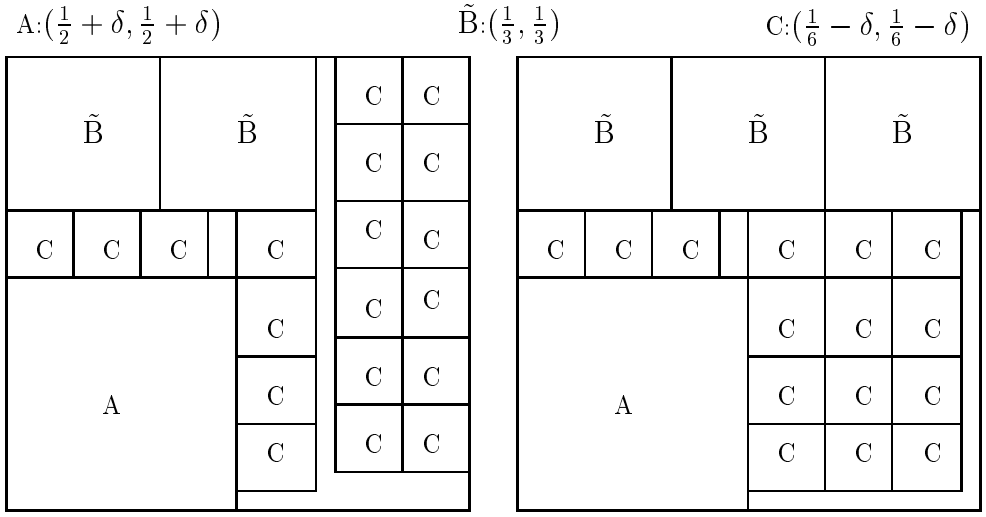
Como a lista L_1 é empacotada pelo Algoritmo UC e as caixas c_i em L_1 são tais que $x_i > \frac{a}{2}$ e $y_i > \frac{b}{2}$, pelo Lema 3.2.14, temos que

$$OPT(L) \geq h_1 \ . \tag{3.14}$$

Das desigualdades (3.13) e (3.14), obtemos que $OPT(L) \geq \max\{h_1, \frac{h_1}{4} + \frac{H}{3}\}$.

Por outro lado, temos também que $C(L) \leq h_1 + H + 7Z$. Assim,

$$\frac{C(L)}{OPT(L)} \leq \frac{h_1 + H}{OPT(L)} + \frac{7Z}{OPT(L)} \leq \frac{h_1 + H}{\max\{h_1, \frac{h_1}{4} + \frac{H}{3}\}} + \frac{7Z}{OPT(L)} \ ,$$



(a) $3n$ níveis deste tipo. (b) n níveis neste tipo.

Figura 3.6: Empacotamento para mostrar a justeza do limite de desempenho do Algoritmo C.

i.e.,

$$C(L) \leq \alpha \cdot OPT(L) + 7Z \ , \tag{3.15}$$

onde $\alpha = \frac{h_1+H}{\max\{h_1, \frac{h_1}{4}+\frac{H}{3}\}}$.

Temos dois casos a analisar.

Caso 1: $h_1 \geq \frac{h_1}{4} + \frac{H}{3}$. Então, $h_1 \geq \frac{4}{9}H$, e portanto,

$$\alpha = \frac{h_1 + H}{h_1} = 1 + \frac{H}{h_1} \leq 1 + \frac{9}{4} = 3,25 \ .$$

Caso 2: $h_1 \leq \frac{h_1}{4} + \frac{H}{3}$. Então $\alpha = \frac{h_1+H}{\frac{h_1}{4}+\frac{H}{3}}$.

Como α é uma função estritamente crescente com relação a h_1 e $h_1 \leq \frac{4}{9}H$, temos que α é máximo quando $h_1 = \frac{4}{9}H$ e, portanto, $\alpha \leq 3,25$.

Em ambos os casos temos que

$$C(L) \leq 3,25 \cdot OPT(L) + 7Z \ .$$

Para mostrar a justeza do limite, considere a seguinte instância L do $PET(1,1)$, $L = I_1 \cup I_2$. A lista I_1 consiste de $4n$ caixas de dimensão $(\frac{1}{2}+\delta, \frac{1}{2}+\delta, 1)$ e a lista I_2

consiste de $90n$ caixas $(c_1, \ldots, c_{90n})$, onde

$$c_i = \begin{cases} \left(\frac{1}{3}, \frac{1}{3}, 1-(i-1)\epsilon\right) & \text{se } i \bmod 10 = 1 \\ \left(\frac{1}{6}-\delta, \frac{1}{6}-\delta, 1-(i-1)\epsilon\right) & \text{se } i \bmod 10 = 2, 3, \ldots, 9 \\ \left(\frac{1}{k}, \frac{1}{k}, 1-(i-1)\epsilon\right) & \text{se } i \bmod 10 = 0, \end{cases}$$

com k suficientemente grande.

Quando o Algoritmo C é aplicado a L, no Passo 1 do algoritmo apenas as listas $L_1 = I_1$ e $L_5 = I_2$ são geradas. Todas as demais listas são vazias.

É claro que a altura do empacotamento $\wp_1$ de I_1 é $4n$.

Vamos então considerar o empacotamento de I_2 gerado pelo algoritmo G_3. Primeiro, G_3 ordena I_2 em ordem não-crescente de altura, não causando alterações em I_2. Então G_3 divide I_2 em sublistas L'_i tal que a área acumulativa em cada sublista não excede $\frac{4}{9}$ (cf. Lema 2.2.14). Fazendo com que

$$\left(\frac{1}{3}\right)^2 + 8\left(\frac{1}{6}-\delta\right)^2 + \left(\frac{1}{k}\right)^2 \leq \frac{4}{9} \qquad \text{e} \qquad \left(\frac{1}{3}\right)^2 + 8\left(\frac{1}{6}-\delta\right)^2 + \left(\frac{1}{k}\right)^2 + \left(\frac{1}{3}\right)^2 > \frac{4}{9},$$

obtemos

$$\frac{1}{k^2} - \frac{8}{3}\delta + 8\delta^2 > 0 \ .$$

Para que esta desigualdade seja válida, basta tomar um δ bem pequeno e $k \geq 3$.

É fácil ver que são $9n$ sublistas $L'_1, L'_2, \ldots, L'_{9n}$, cada uma contendo 10 caixas, i.e.,

$$L_i = (c_{10(i-1)+1}, c_{10(i-1)+2}, \ldots, c_{10i}) \ , \quad \text{para} \quad i = 1, \ldots, 9n \ .$$

Assim, a altura do empacotamento de I_2 é

$$\sum_{i=1}^{9n} z_{10(i-1)+1} = \sum_{i=1}^{9n} (1 - 10(i-1)\epsilon) = 9n - 45n(9n-1)\epsilon \ .$$

Juntando as alturas dos empacotamentos de I_1 e I_2 temos que $C(L) = 13n - 45n(9n-1)\epsilon$.

Considere agora um empacotamento no qual há $(4n+1)$ níveis, dos quais $3n$ níveis são do tipo indicado na Figura 3.6 (a), e n níveis são do tipo indicado na Figura 3.6 (b). Como podemos fazer com que k seja grande o suficiente de forma que o fundo $(\frac{1}{k}, \frac{1}{k})$ seja tão pequeno quanto se queira, podemos empacotar todas as caixas com fundo $(\frac{1}{k}, \frac{1}{k})$ em apenas um nível. Logo,

$$\lim_{\delta,\epsilon \to 0} \frac{C(L)}{OPT(L)} = \frac{13n}{4n+1} < 3,25 \ ,$$

e a razão acima pode ficar tão próxima de 3,25 quanto se queira.

□

3.2.5 Algoritmo CQQ

Li e Cheng [22] mostraram que modificando o Algoritmo C para tratar o caso em que a instância L consiste de caixas com fundo quadrado e $a = b$, pode-se garantir limites de desempenho melhores.

Algoritmo $CQQ(L)$

Entrada: Constante a e uma lista de caixas $L \in \mathcal{Q}(a, a)$.
Saída: Empacotamento $\wp$ de L em $B = (a, a, \infty)$.

1. Particione a lista L em três sublistas, L_1, L_2 e L_3 (veja a Figura 3.7),

$$\begin{aligned} L_1 &= L \bigcap \mathcal{C}\left[\frac{1}{2}, 1\ ;\ \frac{1}{2}, 1\right](a, b)\ , \\ L_2 &= L \bigcap \mathcal{C}\left[\frac{1}{3}, \frac{1}{2}\ ;\ \frac{1}{3}, \frac{1}{2}\right](a, b)\ , \\ L_3 &= L \bigcap \mathcal{C}\left[0, \frac{1}{3}\ ;\ 0, \frac{1}{3}\right](a, b)\ . \end{aligned}$$

2. $\wp_1 \leftarrow UC(L_1)$.
3. $\wp_2 \leftarrow NFDH^x(L_2)$.
4. $\wp_3 \leftarrow GQ_3(L_3)$.
5. $\wp = \wp_1 \| \wp_2 \| \wp_3$.
6. Retorne $\wp$.

fim algoritmo.

Teorema 3.2.16 *Para qualquer instância $L \in \mathcal{Q}(a, b)$, onde nenhuma caixa tem altura maior que Z, temos que*

$$CQQ(L) \leq 2,6875 \cdot OPT(L) + 2Z\ .$$

Mais ainda, este limite é justo.

Dem. A demonstração é análoga à do Teorema 3.2.15. Note que, no Passo 3, o Algoritmo $NFDH^x$ coloca quatro caixas em cada nível, cada um com área de fundo de pelo menos $\frac{a^2}{9}$. Assim, há uma garantia de pelo menos $\frac{4}{9}a^2$ de área de fundo em cada nível, exceto talvez o último.

No Passo 4 é usado o Algoritmo GQ_m, $m = 3$, que garante um mínimo de $\left(\frac{m-1}{m}\right)^2 a^2$ de área de fundo em cada nível, i.e., $\frac{4}{9}a^2$ em cada nível.

Assim, procedendo como no caso do Algoritmo C, obtemos uma razão α tal que

$$\alpha = \frac{h_1 + H}{\max\{h_1, \frac{1}{4}h_1 + \frac{4}{9}H\}} \leq \frac{43}{16} = 2,6875\ .$$

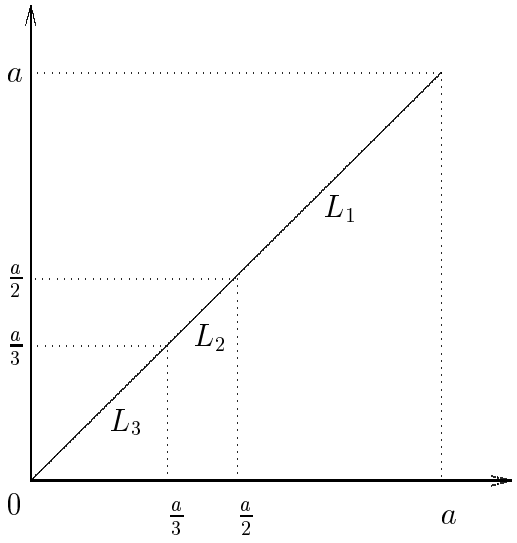
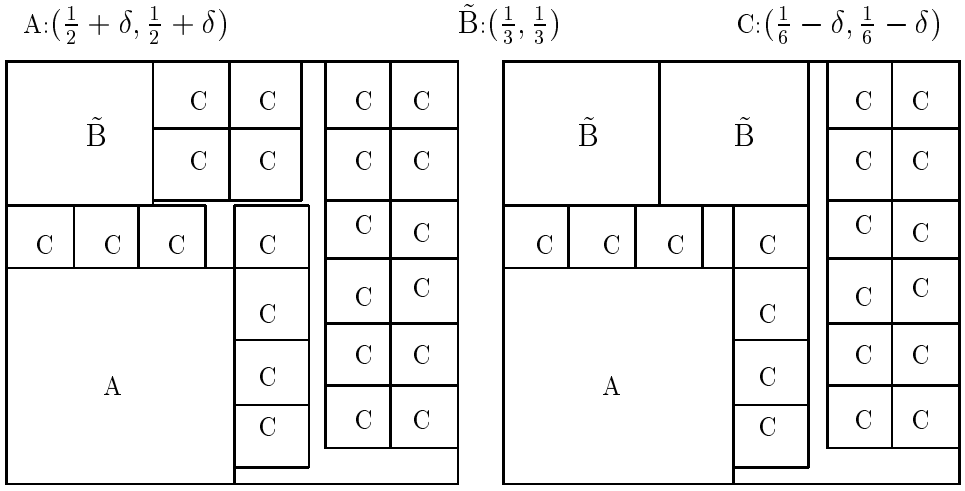


Figura 3.7: Partição da lista L efetuada pelo Algoritmo CQQ.



(a) $5n$ níveis deste tipo.

(b) $11n$ níveis deste tipo.

Figura 3.8: Empacotamentos para mostrar a justeza do limite de desempenho do Algoritmo CQQ.

Para mostrar a justeza do limite mencionado, Li e Cheng construíram uma lista de caixas $L = L_1 \cup L_2$ para o $PET(1,1)$ com $16n$ caixas de tamanho $(\frac{1}{2}+\delta, \frac{1}{2}+\delta, 1)$ em L_1 e $378n$ caixas em L_2, $L_2 = (c_1, \ldots, c_{378n})$, onde

$$c_i = \begin{cases} \left(\frac{1}{3}, \frac{1}{3}, 1-(i-1)\epsilon\right) & \text{se } i \bmod 14 = 1 \\ \left(\frac{1}{6}-\delta, \frac{1}{6}-\delta, 1-(i-1)\epsilon\right) & \text{se } i \bmod 14 = 2, 3, \ldots, 13 \\ \left(\frac{1}{k}, \frac{1}{k}, 1-(i-1)\epsilon\right) & \text{se } i \bmod 14 = 0\ . \end{cases}$$

Escolhendo valores apropriados para δ e k, podemos forçar $L_2 = (c_1, \ldots, c_{378n})$ ser particionado em $27n$ sublistas, cada uma contendo $14n$ caixas. Para isto, impor as duas condições abaixo:

$$\left(\frac{1}{3}\right)^2 + \left(\frac{1}{6}-\delta\right)^2 12 + \frac{1}{k^2} \leq \frac{5}{9} \text{ e} \tag{3.16}$$

$$\left(\frac{1}{3}\right)^2 + \left(\frac{1}{6}-\delta\right)^2 12 + \frac{1}{k^2} + \left(\frac{1}{3}\right)^2 > \frac{5}{9}\ . \tag{3.17}$$

Para satisfazer as inequações (3.16) e (3.17), basta tomarmos δ bem pequeno e $k \geq 3$.

Não é difícil verificar que a altura gerada pelo empacotamento $CQQ(L)$ é

$$CQQ(L) = 43n - 189n(27n-1)\epsilon\ .$$

Por outro lado, há $16n+1$ níveis no empacotamento ótimo, onde $5n$ níveis são do tipo indicado na Figura 3.8(a), $11n$ níveis são do tipo indicado na Figura 3.8(b) e um nível contém as caixas de fundo $(\frac{1}{k}, \frac{1}{k})$. Logo $OPT(L) \leq 16n+1$, donde segue que

$$\frac{CQQ}{OPT(L)} = \frac{43n - 189n(27n-1)\epsilon}{16n+1}\ .$$

Como a razão acima pode se tornar tão próxima de $\frac{43}{16}$ quanto se queira, a prova está completa.

□

3.2.6 Algoritmo $CQ_m,\ m \geq 2$

Li e Cheng [22] também mostraram que a mesma idéia do Algoritmo CQQ pode ser usada para instâncias com caixas de fundo quadrado e pequeno.

Algoritmo $CQ_m(LL),\ m \geq 2$

Entrada: Constantes a e b e uma lista de caixas $L \in \mathcal{Q}_m(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Se $a \geq b$ então

1.1. Divida a lista L em três sublistas L_1, L_2 e L_3, sendo

$$\begin{aligned} L_1 &= L \bigcap \mathcal{C}\left[\frac{1}{m+1}, \frac{1}{m}\ ;\ \frac{1}{m+1}, \frac{1}{m}\right](a,b)\ , \\ L_2 &= L \bigcap \mathcal{C}\left[0, \frac{1}{m+1}\ ;\ \frac{1}{m+1}, \frac{1}{m}\right](a,b)\ , \\ L_3 &= L \bigcap \mathcal{C}\left[0, \frac{1}{m+1}\ ;\ 0, \frac{1}{m+1}\right](a,b)\ . \end{aligned}$$

1.2. $\wp_1 \leftarrow NFDH^x(L_1)$.

1.3. $\wp_2 \leftarrow NFDH^x(L_2)$.

1.4. $\wp_3 \leftarrow GQ_{m+1}(L_3)$.

1.5. $\wp \leftarrow \wp_1 \| \wp_2 \| \wp_3$.

2. Se $a < b$ então construa um empacotamento $\wp$ análogo ao construído no Passo 1.
3. Retorne $\wp$.

fim algoritmo.

Teorema 3.2.17 *Para qualquer lista $L \in \mathcal{Q}_m(a,b)$, $m \geq 2$, onde nenhuma caixa tem altura maior que Z, tem-se que*

$$CQ_m(L) < \left(\frac{m+1}{m}\right)^2 OPT(L) + 3Z\ .$$

Mais ainda, este limite é justo.

Dem. Sejam L_1, L_2 e L_3 as sublistas de L geradas pelo Algoritmo CQ_m. Aplicando o Lema 3.2.10 à lista L_1 obtemos

$$H(\wp_1) \leq \left(\frac{m+1}{m}\right)^2 \frac{V(L_1)}{ab} + Z\ . \tag{3.18}$$

Aplicando o Lema 3.2.11 à lista L_2 obtemos

$$H(\wp_2) \leq \left(\frac{m+1}{m}\right)^2 \frac{V(L_2)}{ab} + Z\ . \tag{3.19}$$

Finalmente, para o empacotamento $\wp_3$ de L_3, obtido pelo algoritmo GQ_{m+1}, temos pelo Lema 3.2.8,

$$H(\wp_3) \leq \left(\frac{m+1}{m}\right)^2 \frac{V(L_3)}{ab} + Z \ . \tag{3.20}$$

De (3.18), (3.19) e (3.20), segue que

$$H(\wp) \leq \left(\frac{m+1}{m}\right)^2 \frac{V(L)}{ab} + 3Z.$$

Usando o Lema 3.2.1 concluimos que

$$H(\wp) \leq \left(\frac{m+1}{m}\right)^2 OPT(L) + 3Z \ .$$

A justeza do limite pode ser provada considerando a instância L_{m+1} construída na prova do Teorema 3.2.9.

□

3.2.7 Algoritmo CQ

No caso especial em que todas as caixas a serem empacotadas têm fundo quadrado e $B = (a, b, \infty)$, não necessariamente com $a = b$, usamos a mesma estratégia de dividir a instância em mais partes para conseguir um melhor resultado.

Algoritmo $CQ(L)$

Entrada: Lista de caixas de fundo quadrado $L \in \mathcal{Q}(a, b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Se $a \geq b$ então

1.1. Particione a lista $L = (c_1, \ldots, c_n)$ em quatro sublistas, $L_1, L_2, \ldots, L_4$ (veja a Figura 3.9).

$$\begin{aligned}
L_1 &= L \bigcap \mathcal{C}\left[0, 1 \ ; \ \frac{1}{2}, 1\right](a, b) \ , \\
L_2 &= L \bigcap \mathcal{C}\left[0, \frac{1}{3} \ ; \ \frac{1}{3}, \frac{1}{2}\right](a, b) \ , \\
L_3 &= L \bigcap \mathcal{C}\left[\frac{1}{3}, \frac{1}{2} \ ; \ \frac{1}{3}, \frac{1}{2}\right](a, b) \ , \\
L_4 &= L \bigcap \mathcal{C}\left[0, \frac{1}{3} \ ; \ 0, \frac{1}{3}\right](a, b) \ .
\end{aligned}$$

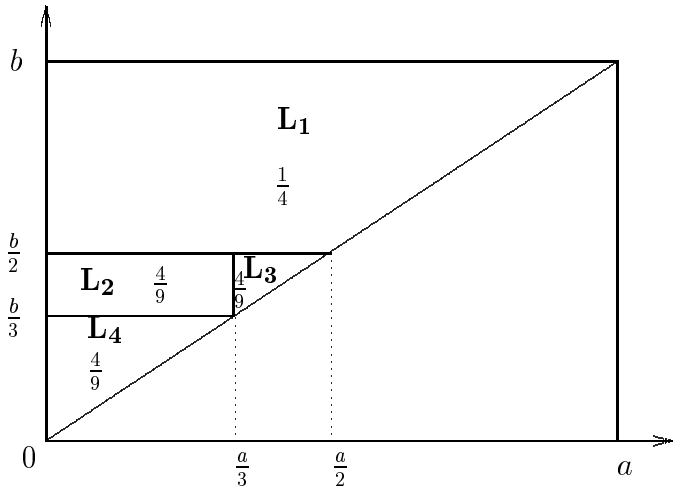


Figura 3.9: Partição da lista L efetuada pelo Algoritmo CQ.

1.2. Gere um empacotamento $\wp_1$ de L_1, procedendo da seguinte maneira.

1.2.1. Divida a lista L_1 em duas sublistas L_1' e L_1'', sendo

$$\begin{aligned} L_1' &= \left\{ c_i \in L_1 : \; x_i y_i > \frac{ab}{4} \right\} , \\ L_1'' &= L_1 \setminus L_1' . \end{aligned}$$

1.2.2. $\wp_1' \leftarrow UC(L_1')$.

1.2.3. $\wp_1'' \leftarrow NFDH^x(L_1'')$.

1.2.4. Construa um empacotamento $\wp_1'''$ de L_1 da seguinte forma.

1.2.4.1. Seja $w(c) = x(c)$ e $h(c) = z(c)$, $\forall c \in L_1$.

1.2.4.2. $\mathcal{B} \leftarrow UD(L_1)$.

1.2.4.3. $(\wp_1'''^x(c), \wp_1'''^y(c), \wp_1'''^z(c)) \leftarrow (\mathcal{B}^w(c), 0, \mathcal{B}^h(c)), \; \forall c \in L_1$.

1.2.5. Seja $\wp_1 \in \{\wp_1' \| \wp_1'', \wp_1'''\}$ tal que $H(\wp_1) = \min\{H(\wp_1' \| \wp_1''), H(\wp_1''')\}$.

1.3. $\wp_i \leftarrow NFDH^x(L_i)$ para $i = 2, 3$.

1.4. $\wp_4 \leftarrow GQ_3(L_4)$.

1.5. $\wp \leftarrow \wp_1 \| \cdots \| \wp_4$.

2. Se $a < b$ então construa um empacotamento análogo $\wp$, procedendo como no Passo 1 (mudando apenas a direção do empacotamento).

3. Retorne $\wp$.

fim algoritmo.

Teorema 3.2.18 *Se L é uma instância do $PET(a,b)$, onde todas as caixas de L têm fundo quadrado e altura no máximo Z, então*

$$CQ(L) \leq 2,796875 \cdot OPT(L) + \frac{77}{8}Z \ .$$

Dem. Pela definição de L_1, duas caixas de L_1 não podem ser colocadas lado a lado na direção do eixo y. Podemos então usar o Algoritmo UD para encontrar um empacotamento $\wp_1'''$ de L_1 e conseguir uma relação com o empacotamento ótimo de L_1. Pelo Teorema 2.2.8, temos que

$$H(\wp_1''') \leq \frac{5}{4}OPT(L_1) + \frac{53}{8}Z \ .$$

Como $L_1 \subseteq L$ e $H(\wp_1) \leq H(\wp_1''')$ obtemos que

$$H(\wp_1) \leq \frac{5}{4}OPT(L) + \frac{53}{8}Z \ .$$

Seja $h_1 = H(\wp_1) - \frac{53}{8}Z$. Substituindo h_1 na desigualdade acima temos que

$$h_1 \leq \frac{5}{4}OPT(L) \ ,$$

i.e.,

$$OPT(L) \geq \frac{4}{5}h_1 \ . \tag{3.21}$$

Como a área de fundo de cada caixa de L_1' é pelo menos $\frac{ab}{4}$, pelo Lema 3.2.13 segue que

$$H(\wp_1') \leq 4\frac{V(L_1')}{ab} \ . \tag{3.22}$$

No empacotamento $\wp_1''$, temos que cada caixa de L_1'' tem comprimento menor que $\frac{a}{2}$. Portanto, $L_1'' \subseteq \mathcal{C}\left[0, \frac{1}{2} \ ; \ \frac{1}{2}, 1\right](a,b)$. Assim, pelo Lema 3.2.11 obtemos que

$$H(\wp_1'') \leq 4\frac{V(L_1'')}{ab} + Z \ . \tag{3.23}$$

Somando as desigualdades (3.22) e (3.23),

$$H(\wp_1' \| \wp_1'') = H(\wp_1') + H(\wp_1'') \leq 4\frac{V(L_1)}{ab} + Z \ .$$

Claramente, $H(\wp_1) \leq H(\wp_1' \| \wp_1'')$. Este fato juntamente com a definição de h_1, implica que

$$h_1 \leq 4\frac{V(L_1)}{ab} + Z - \frac{53}{8}Z \ ,$$

i.e.,

$$h_1 \leq 4\frac{V(L_1)}{ab} \ . \tag{3.24}$$

Note que podemos aplicar o Lema 3.2.11 à lista L_2 obtendo

$$H(\wp_2) \leq \frac{9}{4}\frac{V(L_2)}{ab} + Z \ . \tag{3.25}$$

Aplicando o Lema 3.2.10 à lista L_3 obtemos

$$H(\wp_3) \leq \frac{9}{4}\frac{V(L_3)}{ab} + Z \ . \tag{3.26}$$

O empacotamento de $\wp_4$ é gerado pelo Algoritmo GQ_3, aplicado a L_4. Como $L_4 \in \mathcal{Q}_3(a,b)$, pelo Lema 3.2.8 temos que

$$H(\wp_4) \leq \frac{9}{4}\frac{V(L_4)}{ab} + Z \ . \tag{3.27}$$

Somando as desigualdades (3.25) a (3.27), resulta que

$$\sum_{i=2}^{4} H(\wp_i) \leq \frac{9}{4}\frac{V(L_2 \cup \cdots \cup L_4)}{ab} + 3Z \ .$$

Defina H como $H = \sum_{i=2}^{4} H(\wp_i) - 3Z$. Então, da desigualdade acima temos que

$$H \leq \frac{9}{4}\frac{V(L_2 \cup \cdots \cup L_4)}{ab} \ . \tag{3.28}$$

De (3.24) e (3.28), segue que

$$\frac{V(L)}{ab} = \frac{V(L_1)}{ab} + \frac{V(L_2 \cup \cdots \cup L_4)}{ab} \geq \frac{4}{9}H + \frac{h_1}{4} \ .$$

Como $OPT(L) \geq \frac{V(L)}{ab}$, obtemos da desigualdade acima que

$$OPT(L) \geq \frac{h_1}{4} + \frac{4}{9}H \ . \tag{3.29}$$

Juntando (3.21) e (3.29), temos que

$$OPT(L) \geq \max\left\{\frac{4h_1}{5}, \frac{h_1}{4} + \frac{4H}{9}\right\} \ . \tag{3.30}$$

Por outro lado,

$$H(\wp) = H(\wp_1) + \sum_{i=2}^{4} H(\wp_i) = h_1 + \frac{53}{8}Z + H + 3Z \ ,$$

i.e.,

$$H(\wp) = h_1 + H + \frac{77}{8}Z \ .$$

Então,

$$\begin{aligned} \frac{H(\wp)}{OPT(L)} &= \frac{h_1 + H}{OPT(L)} + \frac{77}{8}\frac{Z}{OPT(L)} \\ &\leq \frac{h_1 + H}{\max\left\{\frac{4}{5}h_1, \frac{h_1}{4} + \frac{4}{9}H\right\}} + \frac{77}{8}\frac{Z}{OPT(L)} \ , \end{aligned}$$

i.e.,

$$H(\wp) \leq \alpha \cdot OPT(L) + \frac{77}{8}Z \ ,$$

onde

$$\alpha = \frac{h_1 + H}{\max\left\{\frac{4}{5}h_1, \frac{h_1}{4} + \frac{4}{9}H\right\}} \ .$$

Para analisar a razão $\alpha = \frac{h_1+H}{\max\left\{\frac{4}{5}h_1, \frac{h_1}{4}+\frac{4}{9}H\right\}}$, vamos considerar dois casos.

Caso 1: $\frac{4}{5}h_1 \geq \frac{h_1}{4} + \frac{4}{9}H$.

Neste caso, $H \leq \frac{99}{80}h_1$, e portanto,

$$\alpha = \frac{h_1 + H}{\frac{4}{5}h_1} \leq \frac{h_1 + \frac{99}{80}h_1}{\frac{4}{5}h_1} = \frac{179}{64} = 2,796875 \ .$$

Caso 2: $\frac{4}{5}h_1 \leq \frac{h_1}{4} + \frac{4}{9}H$.

Neste caso temos que $h_1 \leq \frac{80}{99}H$ e $\alpha = \frac{h_1+H}{\frac{h_1}{4}+\frac{4}{9}H}$.

Como α é uma função estritamente crescente com h_1, α atinge o valor máximo quando $h_1 = \frac{80}{99}H$. Logo,

$$\alpha = \frac{h_1 + H}{\frac{h_1}{4} + \frac{4}{9}H} \leq \frac{\frac{80}{99}H + H}{\frac{1}{4}\frac{80}{99}H + \frac{4}{9}H} = \frac{179}{64} = 2,796875 \ .$$

Tanto no Caso 1 como no Caso 2, temos que $\alpha \leq 2,796875$. Isto completa a prova do teorema.

□

3.2.8 Algoritmo $C_m,\ m \geq 2$

A mesma idéia do Algoritmo CQQ aplicada a instâncias L em $\mathcal{R}_m(a,b)$, também nos dá um bom algoritmo. Note que não há problema em se reparametrizar o problema para que possamos trabalhar com o $PET(1,1)$. Portanto, neste exemplo vamos considerar que o problema foi reparametrizado para o $PET(1,1)$.

Algoritmo $C_m(L)$, $m \geq 2$

Entrada: Lista de caixas $L = (c_1, \ldots, c_n) \in \mathcal{R}_m(1,1)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Divida a lista $L = (c_1, \ldots, c_n)$ em três sublistas,

$$\begin{aligned} L_1 &= L \bigcap \mathcal{C}\left[\frac{1}{m+1}, \frac{1}{m}\ ;\ 0, \frac{1}{m}\right](1,1) \bigcap \mathcal{X}(1,1)\ , \\ L_2 &= L \bigcap \mathcal{C}\left[0, \frac{1}{m}\ ;\ \frac{1}{m+1}, \frac{1}{m}\right](1,1) \bigcap \mathcal{Y}(1,1)\ , \\ L_3 &= L \bigcap \mathcal{C}\left[0, \frac{1}{m+1}\ ;\ 0, \frac{1}{m+1}\right](1,1)\ . \end{aligned}$$

2. $\wp_1 \leftarrow NFDH^y(L_1)$.
3. $\wp_2 \leftarrow NFDH^x(L_2)$.
4. $\wp_3 \leftarrow G_{m+1}(L_3)$.
5. $\wp \leftarrow \wp_1 \| \wp_2 \| \wp_3$.
6. Retorne $\wp$.

fim algoritmo.

Lema 3.2.19 *Para qualquer instância $L \in \mathcal{R}_m(1,1)$, $m \geq 2$, onde nenhuma caixa tem altura maior que Z, tem-se que*

$$C_m(L) \leq \left(\frac{m+1}{m-1}\right) V(L) + 3Z\ .$$

Dem. Sejam L_1, L_2 e L_3 as sublistas de L geradas pelo Algoritmo C_m. Aplicando o Lema 3.2.11 às listas L_1 e L_2, temos que

$$H(\wp_i) \leq \left(\frac{m+1}{m-1}\right) V(L_i) + Z\ ,\quad i = 1, 2\ . \tag{3.31}$$

Pelo Lema 3.2.6, temos que

$$H(\wp_3) \leq \left(\frac{m+1}{m-1}\right) V(L_3) + Z\ . \tag{3.32}$$

Assim, das desigualdades (3.31) e (3.32), temos que

$$H(\wp) \leq \left(\frac{m+1}{m-1}\right) V(L) + 3Z \ .$$

□

Teorema 3.2.20 *Para qualquer instância $L \in \mathcal{R}_m(1,1)$, $m \geq 2$, onde nenhuma caixa tem altura maior que Z, tem-se que*

$$C_m(L) \leq \left(\frac{m+1}{m-1}\right) OPT(L) + 3Z \ .$$

Mais ainda, este limite é justo.

Dem. A desigualdade decorre diretamente da aplicação dos Lemas 3.2.19 e 3.2.1.

A justeza do limite pode ser provada considerando-se a instância L_{m+1} da prova do Teorema 3.2.7.

□

3.2.9 Algoritmo $C_m^*,\ m \geq 2$

Apesar do Algoritmo C_m ter um bom limite de desempenho assintótico (que melhora a medida que m cresce), Li e Cheng não exploraram ao máximo a idéia de subdividir a instância. Apresentamos a seguir um algoritmo que desenvolvemos, que divide a instância em mais partes e tem um limite de desempenho assintótico melhor.

Algoritmo $\mathbf{C_m^*(L)}$, $m \geq 2$

Entrada: Lista de caixas $L \in \mathcal{R}_m(1,1)$.
Saída: Empacotamento $\wp$ de L em $B = (1,1,\infty)$.

1. Particione $L = (c_1, \ldots, c_n)$ em sublistas L_1, L_2, L_3, L_4, assim definidas:

$$\begin{aligned}
L_1 &= L \bigcap \mathcal{C}\left[\frac{1}{m+1}, \frac{1}{m}\ ;\ \frac{1}{m+1}, \frac{1}{m}\right](1,1) \ , \\
L_2 &= L \bigcap \mathcal{C}\left[0, \frac{1}{m+1}\ ;\ \frac{1}{m+1}, \frac{1}{m}\right](1,1) \ , \\
L_3 &= L \bigcap \mathcal{C}\left[\frac{1}{m+1}, \frac{1}{m}\ ;\ 0, \frac{1}{m+1}\right](1,1) \ , \\
L_4 &= L \bigcap \mathcal{C}\left[0, \frac{1}{m+1}\ ;\ 0, \frac{1}{m+1}\right](1,1) \ .
\end{aligned}$$

2. $\wp_1 \leftarrow NFDH(L_1)$.
3. $\wp_2 \leftarrow NFDH^x(L_2)$.
4. $\wp_3 \leftarrow NFDH^y(L_3)$.
5. $\wp_4 \leftarrow C_{m+1}(L_4)$.
6. $\wp \leftarrow \wp_1 \| \wp_2 \| \wp_3 \| \wp_4$.
7. Retorne $\wp$.

fim algoritmo.

Lema 3.2.21 *Para toda lista de caixas $L \in \mathcal{R}_m(1,1)$, $m \geq 2$, onde nenhuma caixa tem altura maior que Z, tem-se que*

$$C_m^*(L) \leq \left(\frac{m+1}{m}\right)^2 V(L) + 6Z \ .$$

Dem. Sejam L_1, L_2, L_3 e L_4 as sublistas de L geradas pelo Algoritmo C_m^*. Aplicando o Lema 3.2.10 à lista L_1 e o Lema 3.2.11 às listas L_2 e L_3, temos que

$$H(\wp_i) \leq \left(\frac{m+1}{m}\right)^2 V(L_i) + Z \ , \quad \text{para} \quad i = 1,2,3 \ . \tag{3.33}$$

Note que $L_4 \in \mathcal{R}_{m+1}(1,1)$. Como L_4 é empacotada pelo Algoritmo C_{m+1}, pelo Lema 3.2.19 temos que

$$H(\wp_4) \leq \left(\frac{m+2}{m}\right) V(L_4) + 3Z \leq \left(\frac{m+1}{m}\right)^2 V(L_4) + 3Z \ . \tag{3.34}$$

De (3.33) e (3.34), segue que

$$H(\wp) \leq \left(\frac{m+1}{m}\right)^2 V(L) + 6Z \ .$$

Isto completa a demonstração do lema.

□

Teorema 3.2.22 *Para toda lista de caixas $L \in \mathcal{R}_m(1,1)$, $m \geq 2$, onde nenhuma caixa tem altura maior que Z, tem-se que*

$$C_m^*(L) \leq \left(\frac{m+1}{m}\right)^2 OPT(L) + 6Z \ .$$

Dem. Segue diretamente da aplicação do Lema 3.2.1 e do Lema 3.2.21.

□

3.2.10 Algoritmo R

Nesta seção descreveremos um algoritmo que desenvolvemos e que aproveita muitas das técnicas usadas no Algoritmo C. Provaremos que este algoritmo tem um limite de desempenho assintótico menor que 3,04904 .

Antes de descrevermos este algoritmo, vamos descrever um outro algoritmo que recebe como entrada duas listas de caixas satisfazendo certas condições, e gera um empacotamento parcial dessas duas listas. Este algoritmo, chamado **Duas Colunas** (**DC**), pode ser informalmente descrito como segue.

Primeiramente o algoritmo determina qual lista de caixas tem a menor soma de alturas e constrói um empacotamento da lista que tem a menor soma, colocando uma caixa em cima da outra (formando uma pilha), começando do fundo. A seguir, vai colocando as caixas da outra lista, uma em cima da outra, começando do fundo, ao lado da pilha da primeira lista. O empacotamento prossegue até que a próxima caixa faça com que a nova pilha ultrapasse ou fique com altura igual à altura da primeira pilha. Neste momento, sem empacotar tal caixa, o processo termina.

Note que, mesmo que as duas listas tenham alturas iguais, uma delas não será totalmente empacotada. Ou seja, o empacotamento é parcial. As duas pilhas geradas ficam uma ao lado da outra, separadas por um plano imaginário, que pode ser perpendicular ao eixo x ou y. Denotaremos por $\mathbf{DC^x}$ (respectivamente $\mathbf{DC^y}$) o algoritmo Duas Colunas que toma este plano perpendicular ao eixo x (respectivamente y). A única restrição sobre as listas de entrada é que estas devem permitir que tal plano possa ser colocado entre o empacotamento. Veja a Figura 3.10.

Uma descrição mais detalhada do algoritmo é dada a seguir.

Algoritmo $DC^x(L_1, L_2)$

Entrada: Lista de caixas $L_1, L_2 \in \mathcal{R}_m(1,1)$ tais que $x(c_i) + x(c_j) < 1$ para toda caixa $c_i \in L_1$ e $c_j \in L_2$.

Saída: Empacotamento parcial $\wp_{1,2}$ de $L_{1,2} \subseteq L_1 \cup L_2$ em $B = (1, 1, \infty)$ e lista de caixas $(L_1 \cup L_2) \setminus L_{1,2}$.

1. Seja $z_1 \leftarrow z(L_1)$, $z_2 \leftarrow z(L_2)$.

2. Se $z_1 \leq z_2$ então construa um empacotamento $\wp_{1,2}$ da seguinte maneira:

2.1 Começando do fundo (da caixa B) coloque todas as caixas de L_1 uma sobre a outra, dispondo-as mais à esquerda.

2.2. Começando do fundo vá colocando as caixas de L_2, uma sobre a outra, no espaço vago que as caixas de L_1 deixaram, dispondo-as mais à direita. Faça isto até que a próxima caixa c a ser empacotada seja tal que a soma das alturas das caixas empacotadas de L_2 mais a altura de c seja maior ou igual a z_1.

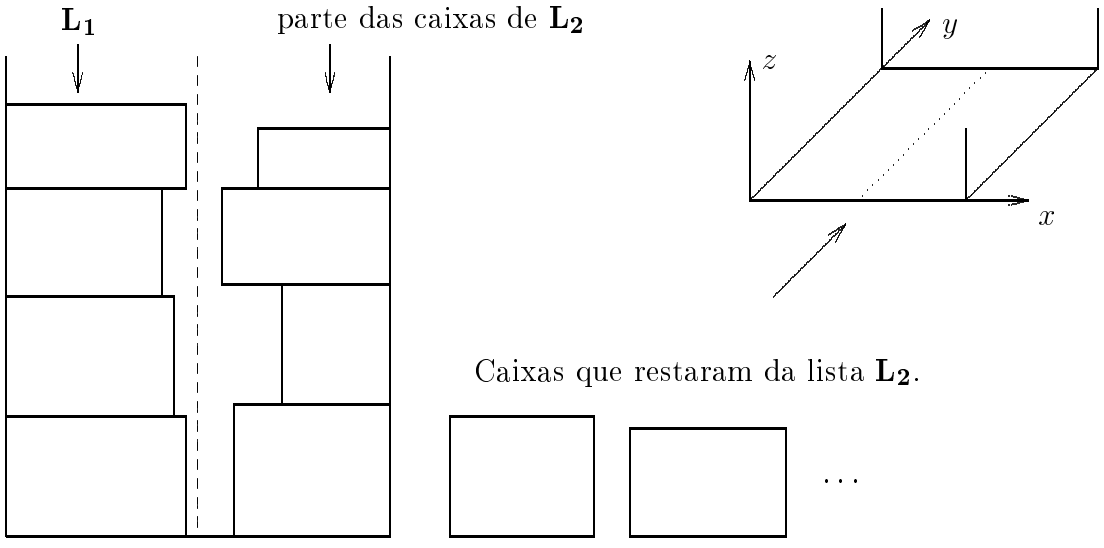


Figura 3.10: Visão frontal do empacotamento gerado por $DC^x(L_1, L_2)$.

Neste momento, sem que a caixa c seja empacotada, pare o processo. Seja L_2' a sublista de L_2 contendo as caixas que não foram empacotadas em $\wp_{1,2}$.

2.3. Retorne $(\wp_{1,2}, L_2')$.

3. Se $z_1 > z_2$ então construa um empacotamento $\wp_{1,2}$ análogo ao do Passo 2, invertendo os papéis de L_1 e L_2.

fim algoritmo.

O Algoritmo DC^y é definido analogamente ao Algoritmo DC^x. Usamos a denominação **DC** para designar tanto DC^x como DC^y. Dizemos que um empacotamento é do **tipo duas colunas** se ele foi gerado pelo Algoritmo DC.

O seguinte lema pode ser observado sobre o Algoritmo DC.

Lema 3.2.23 *Se $(\wp, R)$ é um par resultante da chamada do algoritmo $DC(L_1, L_2)$ e R é uma lista vazia, então L_1 e L_2 devem ser vazias também.*

Lema 3.2.24 *Sejam L_1 e L_2 duas listas de caixas que são sublistas de uma lista L, e seja $\wp$ um empacotamento tipo duas colunas formadas por L_1 e L_2. Se s_1 e s_2 são constantes tais que*

$$S(c) \geq s_1 \quad \forall c \in L_1,$$
$$S(c) \geq s_2 \quad \forall c \in L_2,$$

então

$$H(\wp) < \frac{V(L_1 \cup L_2)}{s_1 + s_2} + Z \ ,$$

onde $Z = \max\{z(c) : c \in L\}$.

Dem. É claro que $H(\wp) = \max\{z(L_1), z(L_2)\}$. Sem perda de generalidade, suponha que $z(L_1) \geq z(L_2)$, i.e., $H(\wp) = z(L_1)$. Então

$$H(\wp) - Z \leq z(L_1) \ . \tag{3.35}$$

Como $z(L_1) - z(L_2) < Z$, substituindo $z(L_1)$ por $H(\wp)$, obtemos

$$H(\wp) - Z \leq z(L_2) \ . \tag{3.36}$$

Como $S(c) \geq s_1$, $\forall c \in L_1$, segue que

$$V(L_1) \geq z(L_1) \cdot s_1 \ .$$

Combinando a desigualdade (3.35), com a desigualdade acima, temos que

$$V(L_1) \geq (H(\wp) - Z) \cdot s_1 \ . \tag{3.37}$$

Fazendo o mesmo para a lista L_2, obtemos

$$V(L_2) \geq (H(\wp) - Z) \cdot s_2 \ . \tag{3.38}$$

Somando as desigualdades (3.37) e (3.38), temos que

$$V(L_1) + V(L_2) \geq (H(\wp) - Z) \cdot (s_1 + s_2) \ ,$$

donde segue que

$$H(\wp) \leq \frac{V(L_1 \cup L_2)}{s_1 + s_2} + Z \ ,$$

provando o lema.

□

Descreveremos a seguir um algoritmo que desenvolvemos, denominado Algoritmo **R**, que faz uso do Algoritmo *DC*.

A estratégia usada para desenvolver este algoritmo foi estudar as partições geradas pelo Algoritmo C, construído por Li e Cheng [23], procurando detectar as partições consideradas *gargalo* do problema. Observamos que a maioria das partições consideradas no

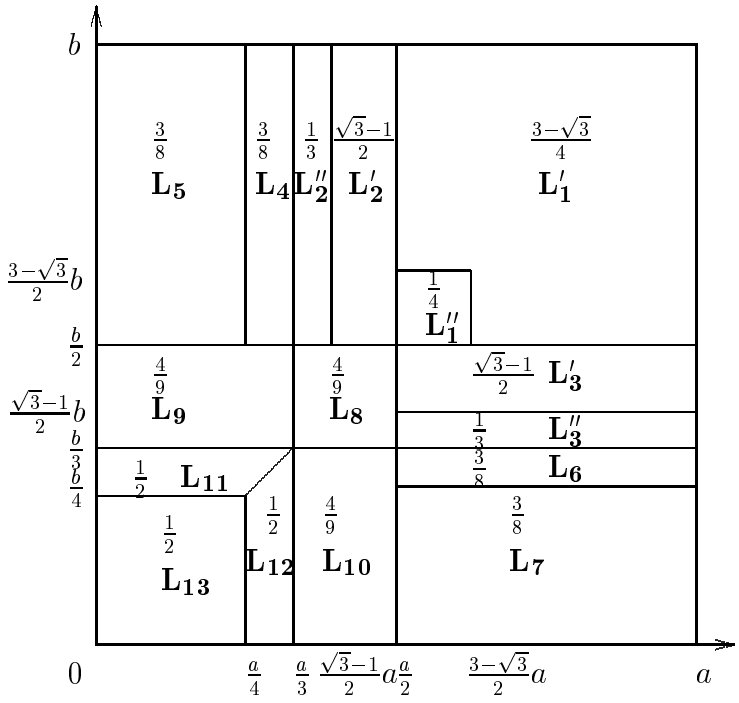


Figura 3.11: Partição da lista L efetuada pelo Algoritmo R.

Algoritmo C, quando submetidas a este algoritmo, geravam empacotamentos com área de fundo de pelo menos $\frac{1}{3}$, exceto uma que tinha área de fundo de pelo menos $\frac{1}{4}$ (veja a Figura 3.5). Notamos que muitas destas partições podiam ser reparticionadas gerando assim empacotamentos com áreas de fundo melhores. Constatamos que o *gargalo* se encontrava em apenas três partições. Mais precisamente, nas sublistas L_1, L_2^x e L_2^y (veja a Figura 3.5). Assim, a estratégia foi usar o Algoritmo DC para combinar estas três sublistas críticas, de forma a obter melhores garantias de área nos empacotamentos destas sublistas.

Algoritmo $R(L)$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Divida a lista L em sublistas $L'_1, L''_1, L'_2, L''_2, L'_3, L''_3, L_4, L_5, \ldots, L_{13}$, como segue (veja a Figura 3.11).

$$L_1'' = L \bigcap \mathcal{C}\left[\tfrac{1}{2}, \tfrac{3-\sqrt{3}}{2}\ ;\ \tfrac{1}{2}, \tfrac{3-\sqrt{3}}{2}\right](a,b), \qquad L_1' = \left(L \bigcap \mathcal{C}\left[\tfrac{1}{2}, 1\ ;\ \tfrac{1}{2}, 1\right](a,b)\right) \setminus L_1'',$$

$$L_2' = L \bigcap \mathcal{C}\left[\tfrac{\sqrt{3}-1}{2}, \tfrac{1}{2}\ ;\ \tfrac{1}{2}, 1\right](a,b), \qquad L_2'' = L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{\sqrt{3}-1}{2}\ ;\ \tfrac{1}{2}, 1\right](a,b),$$

$$L_4 = L \bigcap \mathcal{C}\left[\tfrac{1}{4}, \tfrac{1}{3}\ ;\ \tfrac{1}{2}, 1\right](a,b), \qquad L_5 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ \tfrac{1}{2}, 1\right](a,b),$$

$$L_3' = L \bigcap \mathcal{C}\left[\tfrac{1}{2}, 1\ ;\ \tfrac{\sqrt{3}-1}{2}, \tfrac{1}{2}\right](a,b), \qquad L_3'' = L \bigcap \mathcal{C}\left[\tfrac{1}{2}, 1\ ;\ \tfrac{1}{3}, \tfrac{\sqrt{3}-1}{2}\right](a,b),$$

$$L_6 = L \bigcap \mathcal{C}\left[\tfrac{1}{2}, 1\ ;\ \tfrac{1}{4}, \tfrac{1}{3}\right](a,b), \qquad L_7 = L \bigcap \mathcal{C}\left[\tfrac{1}{2}, 1\ ;\ 0, \tfrac{1}{4}\right](a,b),$$

$$L_8 = L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{1}{2}\ ;\ \tfrac{1}{3}, \tfrac{1}{2}\right](a,b), \qquad L_9 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{3}\ ;\ \tfrac{1}{3}, \tfrac{1}{2}\right](a,b),$$

$$L_{10} = L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{1}{2}\ ;\ 0, \tfrac{1}{3}\right](a,b), \qquad L_{11} = L \bigcap \mathcal{C}\left[0, \tfrac{1}{3}\ ;\ \tfrac{1}{4}, \tfrac{1}{3}\right](a,b) \bigcap \mathcal{Y}(a,b),$$

$$L_{12} = L \bigcap \mathcal{C}\left[\tfrac{1}{4}, \tfrac{1}{3}\ ;\ 0, \tfrac{1}{3}\right](a,b) \bigcap \mathcal{X}(a,b), \qquad L_{13} = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ 0, \tfrac{1}{4}\right](a,b).$$

2. $(\wp_{1,2}, R_{1,2}) \leftarrow DC^x(L_1'', L_2'')$. Seja $L_{1,2}$ a lista das caixas de $\wp_{1,2}$.
3. $(\wp_{1,3}, R_{1,3}) \leftarrow DC^x(L_1'' \setminus L_{1,2}, L_3'')$. Seja $L_{1,3}$ a lista das caixas de $\wp_{1,3}$.
4. Sejam

$$\begin{aligned} L_1 &\leftarrow (L_1' \cup L_1'') \setminus (L_{1,2} \cup L_{1,3}) \ , \\ L_2 &\leftarrow (L_2' \cup L_2'') \setminus L_{1,2} \ , \\ L_3 &\leftarrow (L_3' \cup L_3'') \setminus L_{1,3} \ . \end{aligned}$$

5. $\wp_1 \leftarrow UC(L_1)$.
6. $\wp_i \leftarrow NFDH^x(L_i)$ para $i = 2, 4, 5, 8, 9, 11$.
7. $\wp_i \leftarrow NFDH^y(L_i)$ para $i = 3, 6, 7, 10, 12$.
8. $\wp_{13} \leftarrow G_4(L_{13})$.
9. $\wp \leftarrow \wp_{1,2} \| \wp_{1,3} \| \wp_1 \| \ldots \| \wp_{13}$.
10. Retorne $\wp$.

fim algoritmo.

Teorema 3.2.25 *Para qualquer instância L do $PET(a,b)$, onde nenhuma caixa tem altura maior que Z, tem-se que*

$$R(L) \leq 3,04904 \cdot OPT(L) + 14Z \ .$$

Dem. Primeiramente observe que as listas L_4, L_6 e L_8 são empacotadas pelo Algoritmo $NFDH$. Portanto, pelo Lema 3.2.10, temos

$$H(\wp_i) \leq \frac{8}{3}\frac{V(L_i)}{ab} + Z \ , \quad \text{para } i = 4, 6 \ , \tag{3.39}$$

$$H(\wp_8) \leq \frac{9}{4}\frac{V(L_8)}{ab} + Z \ . \tag{3.40}$$

Observe também que as listas L_5, L_9 e L_{11} são empacotadas pelo Algoritmo $NFDH^x$ e as listas L_7, L_{10} e L_{12} são empacotadas pelo Algoritmo $NFDH^y$. Pelo Lema 3.2.11 segue que

$$H(\wp_i) \leq \frac{8}{3}\frac{V(L_i)}{ab} + Z \ , \quad \text{para } i = 5, 7 \ ; \tag{3.41}$$

$$H(\wp_i) \leq \frac{9}{4}\frac{V(L_i)}{ab} + Z \ , \quad \text{para } i = 9, 10 \ ; \tag{3.42}$$

$$H(\wp_i) \leq 2\frac{V(L_i)}{ab} + Z \ , \quad \text{para } i = 11, 12 \ . \tag{3.43}$$

$$\tag{3.44}$$

A lista L_{13} pertence ao conjunto $R_4(a, b)$ e é empacotada pelo Algoritmo G_4. Portanto, pelo Lema 3.2.6, temos que

$$H(\wp_{13}) \leq 2\frac{V(L_{13})}{ab} + Z \ . \tag{3.45}$$

Observe que todas as caixas de L_2'' e L_3'' têm área de fundo de pelo menos $\frac{1}{6}$, e toda caixa de L_1'' tem área de fundo de pelo menos $\frac{1}{4}$. Logo, pelo Lema 3.2.24,

$$H(\wp_i) \leq \frac{12}{5}\frac{V(L_i)}{ab} + Z \ , \quad \text{para } i \in \{(1, 2), (1, 3)\} \ . \tag{3.46}$$

Juntando as desigualdades (3.39) até (3.46), segue que

$$H(\wp_i) \leq \frac{2}{\sqrt{3} - 1}\frac{V(L_i)}{ab} + Z \ , \quad \text{para } i \in \{(1, 2), (1, 3), 4, \ldots, 13\} \ . \tag{3.47}$$

Daqui em diante iremos dividir a demonstração em três casos, conforme o tipo das caixas da lista $R_{1,3}$ gerada no Passo 3.

Caso 1: $R_{1,3} = \emptyset$.

Pelo Lema 3.2.23, temos que se a lista $R_{1,3}$ é vazia, então $L_1'' \setminus L_{1,2}$ e L_3'' são vazias. Em particular, $L_1'' \subseteq L_{1,2}$, i.e., L_1'' foi totalmente consumida no empacotamento $\wp_{1,2}$ e, portanto, cada caixa da lista L_1 definida no Passo 4 tem área de fundo de pelo menos $\frac{3-\sqrt{3}}{4}$ (pois L_1 torna-se igual a L_1').

Pelo Lema 3.2.13, temos que

$$H(\wp_1) \leq \frac{4}{3 - \sqrt{3}}\frac{V(L_1)}{ab} \ . \tag{3.48}$$

Note que $L_2 \subset \mathcal{C}\left[\frac{1}{3}, \frac{1}{2} \ ; \ \frac{1}{2}, 1\right]$ e $L_3 \subset \mathcal{C}\left[\frac{1}{2}, 1 \ ; \ \frac{1}{3}, \frac{1}{2}\right]$. Assim, aplicando o Lema 3.2.10 às listas L_2 e L_3, temos que

$$H(\wp_i) \leq 3\frac{V(L_i)}{ab} + Z \ , \quad i = 2, 3 \ . \tag{3.49}$$

Das desigualdades (3.47) e (3.49) seque que

$$H(\wp_i) \leq 3\frac{V(L_i)}{ab} + Z \ , \quad \text{para} \quad i \in \{(1,2),(1,3),2,3,\ldots,13\} \ . \tag{3.50}$$

Definindo H como $H = H(\wp_{1,2}) + H(\wp_{1,3}) + H(\wp_2) + \cdots + H(\wp_{13}) - 14Z$, resulta que

$$H \leq 3\frac{V(L_{1,2} \cup L_{1,3} \cup L_2 \cup \cdots \cup L_{13})}{ab} \ . \tag{3.51}$$

Seja $h = H(\wp_1)$. Como $OPT(L) \geq \frac{V(L)}{ab}$, temos pelas desigualdades (3.48) e (3.51) que $OPT(L) \geq \frac{H}{3} + h\left(\frac{3-\sqrt{3}}{4}\right)$.

Pelo Lema 3.2.14 temos também que $OPT(L) \geq h$. Assim,

$$OPT(L) \geq \max\left\{h, \left(\frac{3-\sqrt{3}}{4}\right)h + \frac{H}{3}\right\} \ .$$

Como $H(\wp) = H + h + 14Z$, temos que

$$R(L) \leq \alpha \cdot OPT(L) + 14Z \ , \quad \text{onde} \quad \alpha = \frac{h+H}{\max\left\{h, \left(\frac{3-\sqrt{3}}{4}\right)h + \frac{H}{3}\right\}} \ .$$

Fazendo uma análise do denominador de α, concluimos que $\alpha = \frac{7+3\sqrt{3}}{4} = 3,04903\ldots$.

Caso 2: $\emptyset \neq R_{1,3} \subseteq L_3$.

Como $R_{1,3} \subseteq L_3$, temos que todas as caixas de $L_1'' \setminus L_{1,2}$ foram consumidas no empacotamento $\wp_{1,3}$. Portanto, a lista L_1 definida no Passo 4 só tem caixas de L_1', i.e., toda caixa de L_1' tem área de fundo de pelo menos $\frac{3-\sqrt{3}}{4}$. A continuação da demonstração para este caso é análoga à do Caso 1.

Caso 3: $\emptyset \neq R_{1,3} \subseteq L_1''$.

Neste caso, temos que todas as caixas de L_3'' foram consumidas no empacotamento $\wp_{1,3}$. É fácil ver que $\emptyset \neq R_{1,2} \subseteq L_1''$, e portanto, podemos concluir também que todas as caixas de L_2'' foram consumidas no empacotamento $\wp_{1,2}$, i.e., $L_2 = L_2'$ e $L_3 = L_3'$. Observe que podemos aplicar o Lema 3.2.10 às listas L_2 e L_3 obtendo

$$H(\wp_i) \leq \frac{2}{\sqrt{3}-1}\frac{V(L_i)}{ab} + Z \ , \quad i = 2,3 \ . \tag{3.52}$$

Juntando as desigualdades (3.47) e (3.52), temos

$$H(\wp_i) \leq \frac{2}{\sqrt{3}-1}\frac{V(L_i)}{ab} + Z \ , \quad \text{para} \quad i = \{2, \ldots, 13, (1,2), (1,3)\} \ . \tag{3.53}$$

Como cada caixa de L_1, empacotada pelo Algoritmo UC, tem área de pelo menos $\frac{1}{4}$, temos pelo Lema 3.2.13 que

$$H(\wp_1) \leq 4\frac{V(L_1)}{ab} \ . \tag{3.54}$$

Definindo h e H como no Caso 1, ou seja, $h = H(\wp_1)$ e $H = H(\wp_2) + \cdots + H(\wp_{13}) + H(\wp_{1,2}) + H(\wp_{1,3}) - 14Z$, e usando (3.53), temos que

$$H \leq \frac{2}{\sqrt{3}-1}\frac{V(L_2 \cup \cdots \cup L_{13} \cup L_{1,2} \cup L_{1,3})}{ab} \ . \tag{3.55}$$

De (3.54) e (3.55) segue que

$$\frac{V(L)}{ab} = \frac{V(L_1) + V(L_2 \cup \cdots \cup L_{13} \cup L_{1,2} \cup L_{1,3})}{ab} \geq \frac{h}{4} + \left(\frac{\sqrt{3}-1}{2}\right) H \ .$$

Como $OPT(L) \geq \frac{V(L)}{ab}$ e $OPT(L) \geq h$, temos que $OPT(L) \geq \max\left\{h, \frac{h}{4} + \left(\frac{\sqrt{3}-1}{2}\right) H\right\}$.

Procedendo da mesma forma como fizemos anteriormente, concluimos que

$$R(L) \leq \alpha \cdot OPT(L) + 14Z \ , \quad \text{onde} \quad \alpha = \frac{h+H}{\max\left\{h, \frac{h}{4} + \left(\frac{\sqrt{3}-1}{2}\right) H\right\}}.$$

Analisando α, novamente obtemos que $\alpha \leq \frac{7+3\sqrt{3}}{4} = 3,04903\ldots$.

□

3.2.11 Algoritmo T

Descreveremos nesta seção um algoritmo que desenvolvemos para o $PET(a,b)$, denominado de Algoritmo **T**, cujo limite de desempenho assintótico é menor ou igual a 3.

Este algoritmo aproveita várias idéias já anteriormente exploradas:

⋆ Garantia de área mínima (como no Algoritmo C),
⋆ Combinação de duas listas de caixas (como no Algoritmo R),
⋆ Uso do algoritmo UD (como no Algoritmo CQ da seção anterior).

A idéia deste algoritmo consiste em usar o Algoritmo DC para combinar listas com garantias de área de $\frac{1}{3}$ e $\frac{1}{4}$ de forma que uma destas seja totalmente consumida no empacotamento combinado. Depois, conforme o resultado da combinação, verificar o melhor empacotamento possível.

Sem perda de generalidade, considere o $PET(1,1)$.

Algoritmo $T(L)$

Entrada: Lista de caixas $L \in \mathcal{R}(1,1)$.
Saída: Empacotamento $\wp$ de L em $B = (1,1,\infty)$.

1. Divida a lista L em sublistas L'_1, L''_1, L'''_1, L_2, L'_3, L_4, ..., L_6, L'_7, L_8, ..., L_{15}, como segue (veja a Figura 3.12):

$$L'_1 = L \bigcap \mathcal{C}\left[\tfrac{5}{8},1\ ;\ \tfrac{5}{8},1\right](1,1), \qquad L''_1 = L \bigcap \mathcal{C}\left[\tfrac{1}{2},\tfrac{5}{8}\ ;\ \tfrac{1}{2},1\right](1,1),$$
$$L'''_1 = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ \tfrac{1}{2},\tfrac{5}{8}\right](1,1), \qquad L_2 = L \bigcap \mathcal{C}\left[\tfrac{3}{8},\tfrac{1}{2}\ ;\ \tfrac{1}{2},1\right](1,1),$$
$$L'_3 = L \bigcap \mathcal{C}\left[\tfrac{1}{3},\tfrac{3}{8}\ ;\ \tfrac{1}{2},1\right](1,1), \qquad L_4 = L \bigcap \mathcal{C}\left[\tfrac{1}{4},\tfrac{1}{3}\ ;\ \tfrac{1}{2},1\right](1,1),$$
$$L_5 = L \bigcap \mathcal{C}\left[0,\tfrac{1}{4}\ ;\ \tfrac{1}{2},1\right](1,1), \qquad L_6 = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ \tfrac{3}{8},\tfrac{1}{2}\right](1,1),$$
$$L'_7 = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ \tfrac{1}{3},\tfrac{3}{8}\right](1,1), \qquad L_8 = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ \tfrac{1}{4},\tfrac{1}{3}\right](1,1),$$
$$L_9 = L \bigcap \mathcal{C}\left[\tfrac{1}{2},1\ ;\ 0,\tfrac{1}{4}\right](1,1), \qquad L_{10} = L \bigcap \mathcal{C}\left[\tfrac{1}{3},\tfrac{1}{2}\ ;\ \tfrac{1}{3},\tfrac{1}{2}\right](1,1),$$
$$L_{11} = L \bigcap \mathcal{C}\left[0,\tfrac{1}{3}\ ;\ \tfrac{1}{3},\tfrac{1}{2}\right](1,1), \qquad L_{12} = L \bigcap \mathcal{C}\left[\tfrac{1}{3},\tfrac{1}{2}\ ;\ 0,\tfrac{1}{3}\right](1,1),$$
$$L_{13} = L \bigcap \mathcal{C}\left[0,\tfrac{1}{3}\ ;\ \tfrac{1}{4},\tfrac{1}{3}\right](1,1) \bigcap \mathcal{Y}(1,1), \quad L_{14} = L \bigcap \mathcal{C}\left[\tfrac{1}{4},\tfrac{1}{3}\ ;\ 0,\tfrac{1}{3}\right](1,1) \bigcap \mathcal{X}(1,1),$$
$$L_{15} = L \bigcap \mathcal{C}\left[0,\tfrac{1}{4}\ ;\ 0,\tfrac{1}{4}\right](1,1).$$

2. $(\wp_{1,3}, R_{1,3}) \leftarrow DC^x(L''_1, L'_3)$. Seja $L_{1,3}$ a lista das caixas em $\wp_{1,3}$.
3. $(\wp_{1,7}, R_{1,7}) \leftarrow DC^y(L'''_1 \setminus L_{1,3}, L'_7)$. Seja $L_{1,7}$ a lista das caixas em $\wp_{1,7}$.
4. $L_1 \leftarrow (L'_1 \cup L''_1 \cup L'''_1) \setminus (L_{1,3} \cup L_{1,7})$.
5. $L_3 \leftarrow L'_3 \setminus L_{1,3}$.
6. $L_7 \leftarrow L'_7 \setminus L_{1,7}$.
7. $\wp_1 \leftarrow UC(L_1)$.
8. $\wp_i \leftarrow NFDH^x(L_i)$ para $i = 2, \ldots, 5, 10, 11, 13$.
9. $\wp_i \leftarrow NFDH^y(L_i)$ para $i = 6, \ldots, 9, 12, 14$.
10. $\wp_{15} \leftarrow G_4(L_{15})$.

% Temos quatro casos a considerar conforme os valores de $R_{1,3}$ e $R_{1,7}$.

11. Caso 1: Se $(R_{1,3} \subseteq L''_1)$ e $(R_{1,7} \subseteq L'''_1)$ então

11.1. $\wp \leftarrow \wp_{1,3} \| \wp_{1,7} \| \wp_1 \| \wp_2 \| \wp_4 \| \wp_5 \| \wp_6 \| \wp_8 \| \cdots \| \wp_{15}$.

11.2. Vá para o Passo 15.

12. Caso 2: Se $(R_{1,3} \subseteq L'_3)$ e $(R_{1,7} \subseteq L'''_1)$ então

% Gere um empacotamento $\wp_{1-5}$ de $L_{1-5} = L_1 \cup \cdots \cup L_5$ da seguinte forma:

12.1. Construa um empacotamento $\wp'_{1-5}$ de L_{1-5} da seguinte maneira:

12.1.1. Seja $w(c) = x(c)$ e $h(c) = z(c)$ $\forall c \in L_{1-5}$

12.1.2. $\mathcal{B} \leftarrow UD(L_{1-5})$.

12.1.3. $({\wp'}^x_{1-5}(c), {\wp'}^y_{1-5}(c), {\wp'}^z_{1-5}(c)) \leftarrow (\mathcal{B}^w(c), 0, \mathcal{B}^h(c)),\ \forall c \in L_{1-5}$.

12.2. $\wp''_{1-5} \leftarrow \wp_1 \| \cdots \| \wp_5$.

12.3. Seja $\wp_{1-5} \in \{\wp'_{1-5}, \wp''_{1-5}\}$ tal que $H(\wp_{1-5}) = \min\{H(\wp'_{1-5}), H(\wp''_{1-5})\}$.

12.4. $\wp \leftarrow \wp_{1,3} \| \wp_{1,5} \| \wp_{1-5} \| \wp_6 \| \cdots \| \wp_{15}$.

12.5. Vá para o Passo 15 .

13. Caso 3: Se $(R_{1,3} \subseteq L''_1)$ e $(R_{1,7} \subseteq L_7)$ então proceda analogamente ao Caso 2 (de forma simétrica à reta $((0,0),(1,1))$) .

14. Caso 4: Se $(R_{1,3} \subseteq L_3)$ e $(R_{1,7} \subseteq L_7)$ então

14.1. $\wp \leftarrow \wp_{1,3} \| \wp_{1,5} \| \wp_1 \| \cdots \| \wp_{15}$.

15. Retorne $\wp$.

fim algoritmo.

Teorema 3.2.26 *Para toda lista de caixas L, onde nenhuma caixa tem altura maior que Z, temos que*

$$T(L) \leq 3 \cdot OPT(L) + \frac{141}{8} Z \ .$$

Dem. Os empacotamentos $\wp_{1,3}$ e $\wp_{1,7}$ foram construídos pelo Algoritmo DC. Pelo Lema 3.2.24,

$$H(\wp_{1,3}) \leq \frac{V(L_{1,3})}{\frac{1}{4} + \frac{1}{6}} + Z \ ,$$

e portanto,

$$H(\wp_{1,3}) \leq \frac{12}{5} V(L_{1,3}) + Z \leq \frac{8}{3} V(L_{1,3}) + Z \ . \tag{3.56}$$

Analogamente, para $\wp_{1,7}$ temos

$$H(\wp_{1,7}) \leq \frac{8}{3} V(L_{1,7}) + Z \ . \tag{3.57}$$

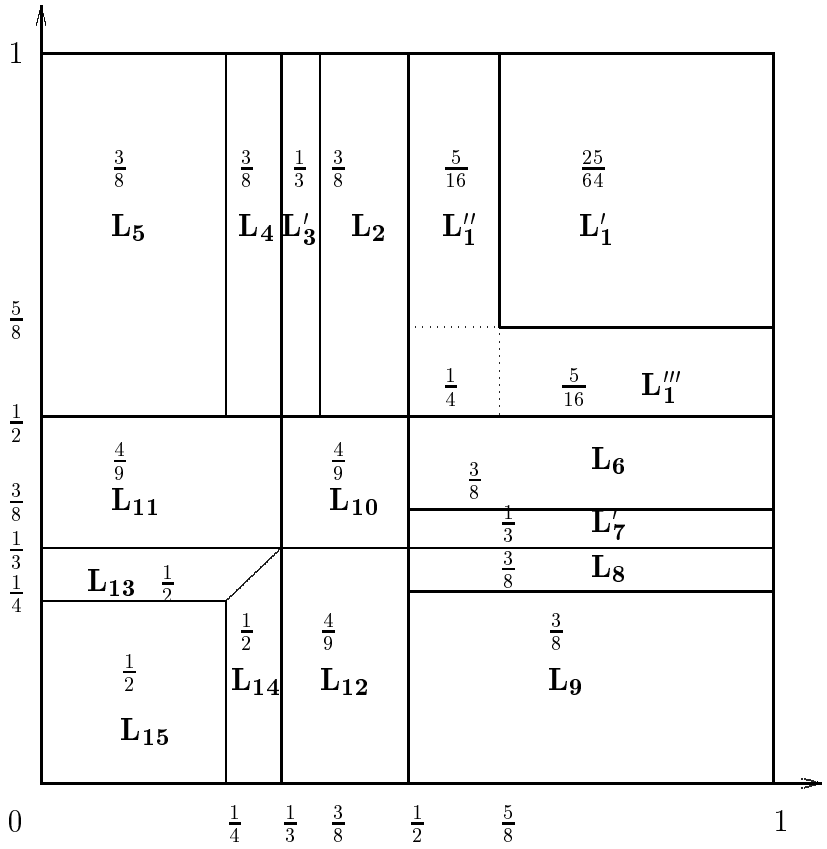


Figura 3.12: Partição da lista L efetuada pelo Algoritmo T.

Da mesma forma como visto anteriormente, valem as seguintes desigualdades:

$$H(\wp_i) \leq 3V(L_i) + Z \quad \text{para } i = 3, 7, \tag{3.58}$$

$$H(\wp_i) \leq \frac{8}{3}V(L_i) + Z \quad \text{para } i = 2, 4, 5, 6, 8, 9, \ldots, 15. \tag{3.59}$$

Vamos analisar cada um dos casos separadamente.

Caso 1: $(R_{1,3} \subseteq L_1'')$ e $(R_{1,7} \subseteq L_1''')$

Como temos $(R_{1,3} \subseteq L_1'')$ e $(R_{1,7} \subseteq L_1''')$, isto significa que todas as caixas de L_3' estão no empacotamento $\wp_{1,3}$ e todas as caixas de L_7' estão em $\wp_{1,7}$. Portanto, $H(\wp_3) = H(\wp_7) = 0$.

Como cada caixa de L_1 tem área de fundo maior ou igual a $\frac{1}{4}$, temos pelo Lema 3.2.13,

$$H(\wp_1) \leq 4V(L_1) \ . \tag{3.60}$$

Juntando as desigualdades (3.56), (3.57) e (3.59), temos

$$H(\wp_i) \leq \frac{8}{3}V(L_i) + Z \quad \text{para } i \in S' = \{(1,3), (1,7), 2, 4, 5, 6, 8, 9, \ldots, 15\} \ . \tag{3.61}$$

Seja $h = H(\wp_i)$ e $H = \sum_{i \in S'} H(\wp_i) - 14Z$. Da definição de h e da desigualdade (3.60) temos que $V(L_1) \geq \frac{1}{4}h$. De (3.61) e da definição de H temos que $\sum_{i \in S'} V(L_i) \geq \frac{3}{8}H$.

Somando estas duas últimas desigualdades, segue que $V(L) \geq \frac{1}{4}h + \frac{3}{8}H$. Como $OPT(L) \geq V(L)$, obtemos

$$OPT(L) \geq \frac{1}{4}h + \frac{3}{8}H \ . \tag{3.62}$$

Note que só podemos colocar as caixas de L_1 uma sobre a outra. Logo,

$$OPT(L) \geq h \ . \tag{3.63}$$

De (3.62) e (3.63), segue que $OPT(L) \geq \max\left\{h, \frac{1}{4}h + \frac{3}{8}H\right\}$. Por outro lado, temos que $H(\wp) = h + H + 14Z$. Portanto,

$$\frac{H(\wp)}{OPT(L)} = \frac{h + H}{OPT(L)} + \frac{14\,Z}{OPT(L)} \leq \frac{h + H}{\max\left\{h, \frac{1}{4}h + \frac{3}{8}H\right\}} + \frac{14\,Z}{OPT(L)}$$

ou seja,

$$H(\wp) \leq \alpha \cdot OPT(L) + 14Z \ ,$$

onde

$$\alpha = \frac{h + H}{\max\left\{h, \frac{1}{4}h + \frac{3}{8}H\right\}} \ .$$

Analisando a razão acima não é difícil concluir que $\alpha \leq 3$.

Caso 2: $(R_{1,3} \subseteq L_3)$ e $(R_{1,7} \subseteq L_1''')$

Pelo Teorema 2.2.8, para o empacotamento $\wp'_{1-5}$ temos que

$$UD(L_{1-5}) \leq \frac{5}{4} \cdot OPT^f(L_{1-5}) + \frac{53}{8}Z \ .$$

Como as caixas não podem ser rotacionadas e todas as caixas de L_{1-5} têm largura maior que $\frac{1}{2}$, temos que $OPT^f(L_{1-5}) = OPT(L_{1-5}) \leq OPT(L)$, i.e.,

$$H(\wp'_{1-5}) \leq \frac{5}{4} \cdot OPT(L) + \frac{53}{8}Z \ .$$

Seja $h = H(\wp_{1-5}) - \frac{53}{8}Z$. Então $h \leq H(\wp_{1-5}) - \frac{53}{8}Z$ e portanto, $h \leq \frac{5}{4}OPT(L)$, i.e.,

$$OPT(L) \geq \frac{4}{5}h \ . \tag{3.64}$$

Como $R_{1,3} \subseteq L_3$, segue que todas as caixas de L_1'' foram consumidas no empacotamento $\wp_{1,3}$ e, portanto, cada caixa de L_1 tem área maior ou igual a $\frac{5}{16}$. Pelo Lema 3.2.13, temos que

$$H(\wp_1) \leq \frac{16}{5}V(L_1) \ . \tag{3.65}$$

De (3.58), (3.59) e (3.65), concluimos que

$$H(\wp''_{1-5}) = \sum_{i=1}^{5} H(\wp_i) \leq \frac{16}{5}V(L_1 \cup \cdots \cup L_5) + 4Z \ . \tag{3.66}$$

Como $h = H(\wp_{1-5}) - \frac{53}{8}Z$ e $H(\wp_{1-5}) \leq H(\wp''_{1-5})$, segue que

$$h \leq \frac{16}{5}V(L_1 \cup \cdots \cup L_5) + 4Z - \frac{53}{8}Z \ ,$$

e portanto,

$$h \leq \frac{16}{5}V(L_1 \cup \cdots \cup L_5) \ . \tag{3.67}$$

Seja $S' = \{(1,3),(1,7),6,8,9,\ldots,15\}$ e $H = \sum_{i \in S'} H(\wp_i) - 11Z$. Então

$$H \leq \sum_{i \in S'} \frac{8}{3}V(L_i) \ ,$$

ou seja,

$$\frac{3}{8}H \leq \sum_{i \in S'} V(L_i) \ . \tag{3.68}$$

Seja $S'' = S' \cup \{1, \ldots, 5\}$.

De (3.67) e (3.68), temos que

$$V(L) = \sum_{i \in S''} V(L_i) \geq \frac{5}{16}h + \frac{3}{8}H \ . \tag{3.69}$$

Como $OPT(L) \geq V(L)$, concluimos que

$$OPT(L) \geq \frac{5}{16}h + \frac{3}{8}H \ . \tag{3.70}$$

As desigualdades (3.64) e (3.70) implicam que

$$OPT(L) \geq \max\left\{\frac{4}{5}h, \frac{5}{16}h + \frac{3}{8}H\right\} \ . \tag{3.71}$$

Por outro lado, $H(\wp) = H(\wp_{1-5}) + \sum_{i \in S'} H(\wp_i)$. Desta desigualdade e das definições de h e H, segue que

$$H(\wp) = \left(h + \frac{53}{8}Z\right) + (H + 11Z) = h + H + \frac{141}{8}Z \ . \tag{3.72}$$

Da mesma forma como procedemos no caso anterior, usando (3.71) e (3.72) obtemos que

$$H(\wp) \leq \alpha \cdot OPT(L) + \frac{141}{8}Z,$$

onde

$$\alpha = \frac{h + H}{\max\left\{\frac{4}{5}h, \frac{5}{16}h + \frac{3}{8}H\right\}} \ .$$

Analisando a razão que define α concluimos que $\alpha \leq \frac{23}{8} = 2.875 < 3$.

Caso 3: $(R_{1,3} \subseteq L_1'')$ e $(R_{1,7} \subseteq L_7)$

A demonstração deste caso é análoga à do Caso 2.

Caso 4: $(R_{1,3} \subseteq L_3)$ e $(R_{1,7} \subseteq L_7)$

Como $(R_{1,3} \subseteq L_3)$ e $(R_{1,7} \subseteq L_7)$, todas as caixas de L_1'' e L_1''' foram consumidas nos empacotamentos $\wp_{1,3}$ e $\wp_{1,7}$. Portanto, todas as caixas de L_1 têm pelo menos $\frac{25}{64}$ de área de fundo. Assim, pelo Lema 3.2.13, temos

$$H(\wp_1) \leq \frac{64}{25}V(L_i) \ . \tag{3.73}$$

Das desigualdades (3.57), (3.58) e (3.59), concluimos que

$$H(\wp_i) \leq 3V(L_i) + Z \quad \text{para } i \in \{(1,3),(1,7),2,\ldots,15\} \ . \tag{3.74}$$

Procedendo como na demonstração do Caso 1, é fácil ver que

$$H(\wp) \leq \alpha \cdot OPT(L) + 16Z,$$

onde

$$\alpha = \frac{h+H}{\max\left\{h, \frac{25}{64}h + \frac{1}{3}H\right\}} \ .$$

Neste caso também, analisando a razão acima concluimos que $\alpha \leq 3$.

Nos quatro casos considerados obtivemos que $H(\wp) \leq 3 \cdot OPT(L) + \frac{141}{8}Z$. Podemos assim considerar completa a prova do teorema.

□

Como esta seção possui muitos algoritmos e a maioria destes partilham das mesmas idéias, construimos um diagrama, figura 3.13, mostrando a evolução dos algoritmos desta seção. Os retângulos representam as principais idéias usadas nos algoritmos e os círculos representam os algoritmos.

Um arco ligando uma idéia X a um algoritmo $\mathcal{A}$ indica que o algoritmo $\mathcal{A}$ faz uso da idéia X. Um arco ligando um algoritmo $\mathcal{A}_1$ a um algoritmo $\mathcal{A}_2$ indica que o algoritmo $\mathcal{A}_2$ aproveita idéias do algoritmo $\mathcal{A}_1$.

Note que os algoritmos G_1, G_m e GQ_m usam como idéia principal a garantia de área em cada nível. Já o Algoritmo C_m é uma evolução do Algoritmo G_m e explora a divisão da instância em mais partes para garantir uma melhor área por nível. Fato análogo ocorre no Algoritmo CQ_m. O Algoritmo C_m^* usa as mesmas idéias do Algoritmo C_m, mas divide a instância em mais partes e garante áreas melhores por nível.

Além das idéias já mencionadas, os algoritmos CQQ e C tratam diferenciadamente as caixas que dão pouca garantia de área. Os algoritmos CQ e R dividem as caixas basicamente em três grupos. Um grupo com caixas com pouca, outro com média e outro com boa garantia de área. O Algoritmo CQ trata os dois primeiros grupos usando um algoritmo de empacotamento bidimensional em faixa e o Algoritmo R trata estes grupos usando o Algoritmo DC (Duas Colunas). Finalmente, o Algoritmo T usa um algoritmo de empacotamento bidimensional em faixa e também o Algoritmo DC para tratar dos grupos críticos.

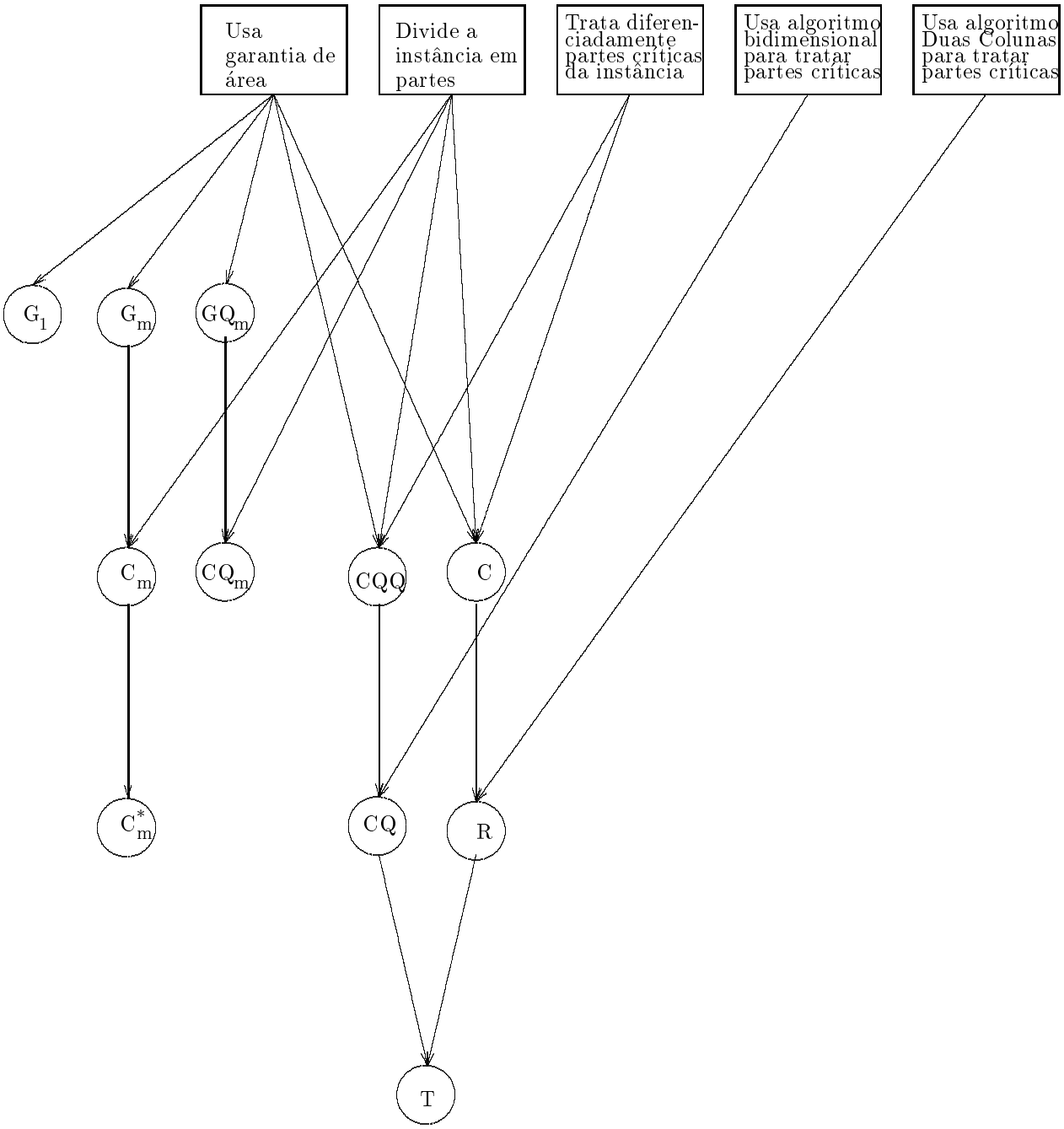


Figura 3.13: Evolução dos algoritmos da seção 3.2 .

3.3 Algoritmos com esquema de arredondamento

Em 1992, Li e Cheng [25] propuseram uma classe de algoritmos de aproximação para o $PET(1,1)$ chamados algoritmos **level strip**. Descreveremos aqui vários desses algoritmos.

A estratégia usada por esses algoritmos consiste em particionar a lista L em sublistas L_1, L_2, ..., de tal forma que em cada sublista L_i as caixas tenham alturas próximas. A partir daí, os algoritmos seguem diferentes esquemas para empacotar cada sublista L_i. Mas todos eles dividem os níveis em faixas, e para a construção dos empacotamentos das caixas nas faixas, usam um algoritmo de empacotamento unidimensional *on-line*.

Esses autores usaram quatro algoritmos de empacotamento unidimensional: NF, FF, BF e H_M (vistos no Capítulo 2).

Vamos supor que $B = (1, 1, \infty)$ e seja T um número tal que $T \geq \max\{z(c) : c \in L\}$. Normalize a altura T para 1 e todas as caixas de L na mesma proporção. Considere L como sendo a lista já normalizada.

Todos os algoritmos desta seção têm como entrada um parâmetro r, $0 < r < 1$, e um empacotamento produzido por estes algoritmos consiste de níveis, sendo que cada nível tem altura r^k, $k \geq 0$, k inteiro.

Toda caixa c em L, tal que $r^{k+1} < z(c) \leq r^k$, é submetida a um arredondamento na altura r^k. O arredondamento pode ser tão pequeno quanto se queira, bastando para isto tomar r suficientemente próximo de 1.

Cada nível é dividido em faixas de largura 1 e comprimentos em um conjunto pré-definido $\{l_1, l_2, \ldots\}$, onde $1 = l_1 > l_2 > \cdots$. Ademais, cada caixa c em L é submetida a um arredondamento no comprimento para l_m quando $l_{m+1} < x(c) \leq l_m$. Uma caixa c com $r^{k+1} < z(c) \leq r^k$ e $l_{m+1} < x(c) \leq l_m$, é empacotada em uma faixa de comprimento l_m em um nível de altura r^k.

Os algoritmos diferem entre si de acordo com

1. o esquema de arredondamento, i.e., o conjunto de possíveis comprimentos para as faixas e a maneira em que um nível é particionado em faixas.

2. o algoritmo para empacotamento unidimensional utilizado para empacotar as caixas na faixa correspondente.

3.3.1 Um primeiro esquema de arredondamento

Veremos aqui uma classe de algoritmos cujo esquema de arredondamento considera o conjunto de comprimentos $\left\{1, \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{2^m}, \ldots\right\}$. Um nível de altura r^k, $k \geq 0$, é chamado um **k-nível**. Uma faixa de comprimento $\frac{1}{2^m}$ em um k-nível é chamada uma **(k, m)-faixa**.

Da mesma forma, uma caixa c tal que $r^{k+1} < z(c) \leq r^k$ e $\frac{1}{2^{m+1}} < x(c) \leq \frac{1}{2^m}$ é chamada uma $(\mathbf{k},\mathbf{m})$**-caixa**. Seja $S_u = \{NF, FF, BF, H_M\}$. Para cada $\mathcal{A} \in S_u$, definiremos um algoritmo chamado $\mathcal{A}\mathbf{LS_r}$. Este algoritmo empacota todas as (k,m)-caixas em (k,m)-faixas usando o algoritmo unidimensional $\mathcal{A}$. Damos a seguir, uma descrição mais formal do Algoritmo $\mathcal{A}LS_r$.

Algoritmo $\mathcal{A}LS_r(L)$**,** $\mathcal{A} \in S_u$**,** $0 < r < 1$

Entrada: Lista de caixas $L = (c_1, \ldots, c_n) \in \mathcal{R}(1,1)$.
Saída: Empacotamento $\wp$ de L em $B = (1, 1, \infty)$.

1. Para cada caixa $c \in L$, faça:

1.1. Sejam k e m inteiros tal que c é uma (k,m)-caixa.

1.2. Seja S o conjunto das (k,m)-faixas criadas até o momento.

1.3. Se o Algoritmo $\mathcal{A}$ consegue empacotar a caixa c em uma das faixas de S, então empacote-a.

1.4. Caso não consiga, procure uma (k,m')-faixa tal que $m' < m$ e m' é o maior possível.

Caso encontre: Particione a (k,m')-faixa em novas faixas de comprimento $2^{-(m'+1)}$, $2^{-(m'+2)}$, ..., $2^{-(m-1)}$, 2^{-m} e 2^{-m} e empacote a caixa c em uma das duas (k,m)-faixas criadas.

Caso não encontre: Crie um novo k-nível e particione este nível em novas faixas de comprimentos $\frac{1}{2}, \frac{1}{4}, \ldots, 2^{-(m-1)}, 2^{-m}$ e 2^{-m} e empacote a caixa c em uma das duas (k,m)-faixas criadas.

2. Seja $\wp$ o empacotamento construído no Passo 1.
3. Retorne $\wp$.

fim algoritmo.

As constantes $U_{\mathcal{A}}$ e $C_{\mathcal{A}}$, bem como as funções peso $W_{\mathcal{A}}$ que serão mencionadas a seguir, são aquelas definidas na Seção 2.1 do capítulo anterior.

Teorema 3.3.1 *Para qualquer constante r, $0 < r < 1$, e qualquer lista de caixas L na qual nenhuma caixa tem altura maior que Z, temos que*

$$\mathcal{A}LS_r(L) < \frac{2U_{\mathcal{A}}}{r} OPT(L) + \frac{1 + 2C_{\mathcal{A}}}{r(1-r)} Z \ .$$

Mais ainda, o limite $\frac{2U_{\mathcal{A}}}{r}$ é justo se $U_{\mathcal{A}}$ é um limite justo para o Algoritmo $\mathcal{A}$.

Dem. Defina o volume de uma caixa c associada à função peso $W_{\mathcal{A}}$ como sendo $\overline{V}(c) = z(c)x(c)W_{\mathcal{A}}(y(c))$. Seja $\overline{V}(L) = \sum_{c \in L} \overline{V}(c)$. Então vale o seguinte resultado.

- $\overline{V}(L) \leq U_{\mathcal{A}} OPT(L)$.

De fato,

$$\begin{aligned}\overline{V}(L) &= \sum_{c\in L} z(c)x(x)W_{\mathcal{A}}(y(c)) \leq \sum_{c\in L} z(c)x(c)U_{\mathcal{A}}y(c) \\ &\leq U_{\mathcal{A}} \sum_{c\in L} z(c)x(x)y(c) \leq U_{\mathcal{A}} \cdot V(L) \leq U_{\mathcal{A}} \cdot OPT(L) \ .\end{aligned}$$

Note que enquanto o empacotamento da lista L é gerado, L é dividido em sublistas L_0, $L_1, \ldots, L_k, \ldots$, onde $L_k = \{c \in L : r^{k+1} < z(c) \leq r^k\}$. Posteriormente, cada sublista L_k é dividida em sublistas $L_{k,0}, L_{k,1}, \ldots, L_{k,m}, \ldots$, onde $L_{k,m} = \{c \in L_k : \frac{1}{2^{m+1}} < x(c) \leq \frac{1}{2^m}\}$, i.e., a lista $L_{k,m}$ contém todas as (k,m)-caixas.

Seja $\overline{V}_k = \overline{V}(L_k)$. Como todas as (k,m)-caixas têm altura maior que r^{k+1} e comprimento maior que $\frac{1}{2^{m+1}}$, temos que

$$\begin{aligned}\overline{V}_k &> r^{k+1} \sum_{c\in L_k} x(c)W_{\mathcal{A}}(y(c)) \\ &= r^{k+1} \sum_{m\geq 0} \sum_{c\in L_{k,m}} x(c)W_{\mathcal{A}}(y(c)) \\ &> r^{k+1} \sum_{m\geq 0} \left(\frac{1}{2^{m+1}} \sum_{c\in L_{k,m}} W_{\mathcal{A}}(y(c)) \right) \ .\end{aligned}$$

Seja $n_{k,m}$ o número de (k,m)-faixas não vazias em todos os k-níveis. Note que todas as caixas em $L_{k,m}$ são empacotadas em $n_{k,m}$ (k,m)-faixas, usando o Algoritmo $\mathcal{A}$. Pelo Lema 2.1.2, $\sum_{c\in L_{k,m}} W_{\mathcal{A}}(y(c)) \geq n_{k,m} - C_{\mathcal{A}}$. Portanto,

$$\overline{V}_k > \frac{r^{k+1}}{2} \sum_{m\geq 0} \frac{1}{2^m}(n_{k,m} - C_{\mathcal{A}}) = \frac{r^{k+1}}{2} \left(\sum_{m\geq 0} \frac{n_{k,m}}{2^m} - 2C_{\mathcal{A}} \right) \ .$$

Seja N_k o número de k-níveis. Então vale a seguinte desigualdade:

- $\sum_{m\geq 0} n_{k,m} \frac{1}{2^m} > N_k - 1$.

Para provar este fato, note que para qualquer $k \geq 0$, todas as faixas vazias em todos os k-níveis são de comprimentos diferentes, i.e., há no máximo uma (k,m)-faixa vazia para cada m. Como não há uma $(k,0)$-faixa vazia, o comprimento total de todas as faixas vazias é menor que $\sum_{m\geq 1} 2^{-m} = 1$. Como $\sum_{m\geq 0} n_{k,m} 2^{-m}$ é o comprimento total de todas as faixas não vazias em todos os k-níveis, o fato segue.

Isto nos dá

$$\overline{V}_k > \frac{r^{k+1}}{2}(N_k - (1+2C_{\mathcal{A}})) \ .$$

Seja Z a altura da caixa mais alta em L e p tal que $r^{p+1} < Z \leq r^p$. Claramente, $\mathcal{A}LS_r(L) = \sum_{k\geq p} r^k N_k$. Portanto, temos que

$$\begin{aligned}
\overline{V}(L) &= \sum_{k\geq p} \overline{V}_k \\
&> \sum_{k\geq p} \frac{r^{k+1}}{2}(N_k - (1+2C_{\mathcal{A}})) \\
&= \frac{r}{2}\left(\mathcal{A}LS_r(L) - \frac{(1+2C_{\mathcal{A}})r^p}{1-r}\right) \ .
\end{aligned}$$

Conseqüentemente,

$$\mathcal{A}LS_r(L) < \frac{2}{r}\overline{V}(L) + \frac{(1+2C_{\mathcal{A}})r^p}{1-r} < \frac{2U_{\mathcal{A}}}{r}OPT(L) + \frac{(1+2C_{\mathcal{A}})}{r(1-r)}Z \ .$$

O seguinte exemplo demonstra a justeza do limite $\frac{2U_{\mathcal{A}}}{r}$, se $U_{\mathcal{A}}$ é um limite justo para o Algoritmo $\mathcal{A}$. Seja $L' = (p_1, p_2, \ldots, p_n)$ uma lista de itens garantida pelo Lema 2.1.3. A partir de L' defina uma lista de caixas $L = (c_1, \ldots, c_n)$, onde $c_i = (2^{-a} + 2^{-2a}, p_i, r + \xi)$, $1 \leq i \leq n$, onde a é um inteiro grande e ξ é um número bem pequeno. Claramente, no empacotamento produzido pelo Algoritmo $\mathcal{A}LS_r$, há pelo menos $\mathcal{A}(L')$ faixas de altura 1 e comprimento 2^{a-1}. Denote por N' o menor número de faixas usadas para empacotar L'. Como $\frac{1}{2^a} + \frac{1}{2^{2a}} \in \left(\frac{1}{2^a}, \frac{1}{2^{a-1}}\right]$, temos que $\mathcal{A}LS_r(L) \geq \left\lceil \frac{U_{\mathcal{A}}N' - D_{\mathcal{A}}}{2^{a-1}} \right\rceil \geq \frac{U_{\mathcal{A}}N' - D_{\mathcal{A}}}{2^{a-1}}$.

Considere outro empacotamento no qual N' faixas de altura $r+\xi$, comprimento $2^{-a} + 2^{-2a}$ e largura 1 são empacotadas.

Há pelo menos $\lfloor \frac{1}{2^{-a}+2^{-2a}} \rfloor$ faixas em cada nível. Como $\lfloor \frac{1}{2^{-a}+2^{-2a}} \rfloor > \frac{1}{2^{-a}+2^{-2a}} - 1 = \frac{2^a - 2^{-a} - 1}{1+2^{-a}}$, a altura deste empacotamento é no máximo

$$\left\lceil \frac{N'}{\left(\frac{2^a - 2^{-a} - 1}{1+2^{-a}}\right)} \right\rceil (r+\xi) < \frac{r(1+2^{-a})}{2^a - 2^{-a} - 1}N' + C \ ,$$

onde C é uma constante.

Portanto,

$$\frac{\mathcal{A}LS_r(L)}{OPT(L)} \geq \frac{2(U_{\mathcal{A}}N' - D_{\mathcal{A}})}{r(1+2^{-a})N' + C(2^a - 2^{-a} - 1)} \frac{2^a - 2^{-a} - 1}{2^a} \ .$$

Quando $N' \to \infty$, a desigualdade se aproxima de

$$\frac{2U_{\mathcal{A}}}{r}\frac{2^a - 2^{-a} - 1}{2^a(1+2^{-a})} ,$$

e este valor pode ser tomado arbitrariamente próximo de $\frac{2U_{\mathcal{A}}}{r}$, escolhendo para a um valor suficientemente grande.

3.3.2 Um esquema de arredondamento melhorado

O esquema de arredondamento anterior pode causar uma perda de 50% de espaço no pior caso. Refinando o esquema de arredondamento, pode-se melhorar o Algoritmo $\mathcal{A}LS_r$. Consideraremos aqui um esquema que permite comprimentos de faixa com valores 2^{-m} ou $\frac{1}{2}2^{-m}$, $m \geq 0$. Ou seja, neste caso, o conjunto de comprimentos de faixa possíveis é

$$\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{6}, \frac{1}{8}, \frac{1}{12}, \frac{1}{16}, \ldots, \frac{1}{2^m}, \frac{1}{3\cdot 2^{m-1}}, \frac{1}{2^{m+1}}, \ldots\right\} .$$

Vamos denotar por $\mathcal{A}\mathbf{LS}^*_{\mathbf{r}}$ o algoritmo que usa este novo esquema de arredondamento. Neste algoritmo, como no algoritmo anterior, durante o empacotamento da lista L, esta é dividida em sublistas $L_1, L_2, \ldots, L_k, \ldots$, de acordo com as alturas das caixas. Além disso, cada lista L_k, $k \geq 0$, é posteriormente dividida em sublistas L^0_k, L^1_{k,m_1}, $m_1 \geq 1$, e L^2_{k,m_2}, $m_2 \geq 0$, definidas a seguir.

$$\begin{aligned}
L^0_k &= \left\{c \in L_k : \ x(c) > \frac{1}{2}\right\} , \\
L^1_k &= \bigcup_{m_1 \geq 1} L^1_{k,m_1} , \quad \text{onde } L^1_{k,m_1} = \left\{c \in L_k : \ \frac{1}{3\cdot 2^{m_1-1}} < x(c) \leq \frac{1}{2^{m_1}}\right\} , \\
L^2_k &= \bigcup_{m_1 \geq 0} L^2_{k,m_2} , \quad \text{onde } L^2_{k,m_2} = \left\{c \in L_k : \ \frac{1}{2^{m_2+2}} < x(c) \leq \frac{1}{3\cdot 2^{m_2}}\right\} .
\end{aligned}$$

Todos os k-níveis são particionados em três classes: 0, 1 e 2. Um nível na classe i é chamado **$(\mathbf{k}, \mathbf{i})$-nível**. Cada $(k,0)$-nível tem apenas uma faixa chamada **$(\mathbf{k}, \mathbf{0})$-faixa**, que contém caixas de L^0_k. Faixas de comprimento 2^{-m_1} ($m_1 \geq 1$), chamadas **$(\mathbf{k}, \mathbf{m_1}, \mathbf{1})$-faixas**, são empacotadas em $(k,1)$-níveis; e faixas de comprimento $\frac{1}{3}2^{-m}$ ($m_2 \geq 0$), chamadas **$(\mathbf{k}, \mathbf{m_2}, \mathbf{2})$-faixas**, são empacotadas em $(k,2)$-níveis. Caixas em L^0_k (L^1_{k,m_1}, L^2_{k,m_2}) chamadas $(k,0)$-caixas, ($(k,m_1,1)$-caixas, $(k,m_2,2)$-caixas) são empacotadas em $(k,0)$-faixas ($(k,m_1,1)$-faixas, $(k,m_2,2)$-faixas) usando o Algoritmo $\mathcal{A}$. A descrição do algoritmo $\mathcal{A}LS^*_r$ é dada a seguir.

Algoritmo $\mathcal{A}LS_r^*(L)$, $\mathcal{A} \in S_u$, $0 < r < 1$

Entrada: Lista de caixas $L = (c_1, \ldots, c_n) \in \mathcal{R}(1,1)$.
Saída: Empacotamento $\wp$ de L em $B = (1, 1, \infty)$.

1. Para cada caixa $c \in L$, faça:

1.1. Seja k tal que $r^{k+1} < x(c) \leq r^k$.

1.2. Se $c \in L_k^0$ então

1.2.1. Empacote a caixa c usando o Algoritmo $\mathcal{A}$ sobre as $(k,0)$-faixas.

1.2.2. Caso não consiga, crie uma nova $(k,0)$-faixa em um novo k-nível e empacote a caixa c usando $\mathcal{A}$ sobre a nova faixa.

1.3. Se $c \in L_k^1$ então

1.3.1. Seja m_1 tal que $\frac{1}{3 \cdot 2^{m_1 - 1}} < x(c) \leq \frac{1}{2^{m_1}}$.

1.3.2. Empacote a caixa c usando $\mathcal{A}$ sobre as $(k, m_1, 1)$-faixas.

1.3.3. Caso não consiga, encontre uma nova $(k, m_1', 1)$-faixa vazia (se necessário crie um novo $(k,1)$-nível) tal que $m_1' < m_1$, sendo m_1' o maior possível. Divida esta faixa em faixas de comprimentos $2^{-(m_1'+1)}, 2^{m_1'+2}, \ldots, 2^{-(m_1-1)}$, 2^{-m_1} e 2^{-m_1} e empacote a caixa c em uma das duas $(k, m_1, 1)$-faixas.

1.4. Se $c \in L_k^2$ então

1.4.1. Seja m_2 tal que $\frac{1}{2^{m_2+2}} < x(c) \leq \frac{1}{3 \cdot 2^{m_2}}$.

1.4.2. Empacote a caixa c usando $\mathcal{A}$ sobre as $(k, m_2, 2)$-faixas.

1.4.3. Caso não consiga, encontre uma $(k, m_2', 2)$-faixa vazia (se necessário crie um novo $(k,2)$-nível, criando inicialmente faixas de comprimento $\frac{1}{3}$) tal que $m_2' < m_2$, sendo m_2' o maior possível. Divida esta faixa em novas faixas de comprimentos $\frac{1}{3}2^{-(m_2'+1)}$, $\frac{1}{3}2^{m_2'+2}$, $\ldots$, $\frac{1}{3}2^{-(m_2-1)}$, $\frac{1}{3}2^{-m_2}$ e $\frac{1}{3}2^{-m_2}$ e empacote a caixa c em uma das duas $(k, m_2, 2)$-faixas.

2. Seja $\wp$ o empacotamento construído no Passo 1.
3. Retorne $\wp$.

fim algoritmo.

Teorema 3.3.2 *Para qualquer constante r, $0 < r < 1$, e qualquer lista de caixas L na qual nenhuma caixa tem altura maior que Z, temos que*

$$\mathcal{A}LS_r^*(L) < \frac{1,75U_{\mathcal{A}}}{r} OPT(L) + \left(2 + \frac{8}{3}C_{\mathcal{A}}\right) \frac{Z}{r(1-r)} \ .$$

Mais ainda, o limite $\frac{1,75U_{\mathcal{A}}}{r}$ é justo se $U_{\mathcal{A}}$ é um limite justo para o Algoritmo $\mathcal{A}$.

Dem. Seja N_k^i o número de (k,i)-níveis e seja $\overline{V}_k^i = \overline{V}(L_k^i)$, onde $i = 0, 1, 2$. Seja n_{k,n_j}^j o número de (k, m_j, j)-faixas para empacotar a lista L_{k,m_j}^j, e $\overline{V}_{k,m_j}^j = \overline{V}(L_{k,m_j}^j)$, $j = 1, 2$.

Note que o número de $(k,0)$-faixas é o mesmo que o número de $(k,0)$-níveis. Portanto,

$$\begin{aligned}
\overline{V}_k^0 &= \sum_{c \in L_k^0} z(c)x(c)W_{\mathcal{A}}(y(c)) \\
&> \frac{r^{k+1}}{2} \sum_{c \in L_k^0} W_{\mathcal{A}}(y(c)) \\
&\geq \frac{r^{k+1}}{2}(N_k^0 - C_{\mathcal{A}}) \ .
\end{aligned}$$

Para L_k^1, temos que

$$\begin{aligned}
\overline{V}_k^1 &= \sum_{m_1 \geq 1} \overline{V}_{k,m_1} \\
&= \sum_{m_1 \geq 1} \sum_{c \in L_{k,m_1}^1} z(c)x(c)W_{\mathcal{A}}(y(c)) \\
&> \sum_{m_1 \geq 1} \left(r^{k+1} \frac{1}{3 \cdot 2^{m_1 - 1}} \sum_{c \in L_{k,m_1}^1} W_{\mathcal{A}}(y(c)) \right) \\
&\geq \frac{2}{3} r^{k+1} \sum_{m_1 \geq 1} \left(n_{k,m_1}^1 - C_{\mathcal{A}} \right) 2^{-m_1} \ .
\end{aligned}$$

Como podemos colocar 2^{m_1} $(k, m_1, 1)$-faixas em um nível, segue que

$$\overline{V}_k^1 > \frac{2}{3} r^{k+1}(N_k^1 - (1 + C_{\mathcal{A}})) \ .$$

Para L_k^2, note que todas as faixas vazias em todos os $(k,2)$-níveis têm comprimentos diferentes, exceto quando $m_2 = 0$. Sendo $m_2 = 0$ pode haver duas faixas vazias, pois no nível com $m_2 = 0$, pode haver duas faixas de comprimento $\frac{1}{3}$ e a outra de $\frac{1}{3}$ subdividida. Portanto, o comprimento total de todas as faixas vazias em todos os $(k,2)$-níveis é menor que $\frac{1}{3} + \sum_{m_2 \geq 0} \frac{1}{3} 2^{-m_2} = 1$, i.e., $\sum_{m_2 \geq 0} n_{k,m_2}^2 \frac{1}{3} 2^{-m_2} > N_k^2 - 1$. Assim,

$$\begin{aligned}
\overline{V}_k^2 &= \sum_{m_2 \geq 0} \overline{V}_{k,m_2}^2 \\
&> \sum_{m_2 \geq 0} \left(r^{k+1} \frac{1}{2^{m_2+2}} \sum_{c \in L_{k,m_2}^2} W_{\mathcal{A}}(y(c)) \right) \\
&\geq \frac{3}{4} r^{k+1} \sum_{m_2 \geq 0} \left(n_{k,m_2}^2 - C_{\mathcal{A}} \right) \frac{1}{3 \cdot 2^{m_2}}
\end{aligned}$$

$$
\begin{aligned}
&> \frac{3}{4} r^{k+1} \left[N_k^2 - \left(1 + \frac{2}{3} C_{\mathcal{A}} \right) \right] \\
&> \frac{2}{3} r^{k+1} \left[N_k^2 - \left(1 + \frac{2}{3} C_{\mathcal{A}} \right) \right] \ .
\end{aligned}
$$

Portanto, se $r^{p+1} < Z \leq r^p$, usando um pouco de manipulação algébrica, obtemos

$$
\begin{aligned}
U_{\mathcal{A}} \cdot OPT(L) &\geq \overline{V}(L) \\
&= \sum_{k \geq p} \overline{V}_k \\
&= \sum_{k \geq p} \left(\overline{V}_k^0 + \overline{V}_k^1 + \overline{V}_k^2 \right) \\
&> \sum_{k \geq p} \left[\frac{r}{2} r^k (N_k^0 - C_{\mathcal{A}}) + \frac{2r}{3} r^k (N_k^1 - (1 + C_{\mathcal{A}})) + \frac{2r}{3} r^k (N_k^2 - (1 + \frac{2}{3} C_{\mathcal{A}})) \right] \\
&= r \left(\frac{1}{2} \mathcal{H}_1 + \frac{2}{3} \mathcal{H}_2 \right),
\end{aligned}
$$

onde

$$
\mathcal{H}_1 = \sum_{k \geq p} r^k (N_k^0 - C_{\mathcal{A}}) \quad \text{e} \quad \mathcal{H}_2 = \sum_{k \geq p} r^k \left((N_k^1 - (1 + C_{\mathcal{A}})) + (N_k^2 - (1 + \frac{2}{3} C_{\mathcal{A}})) \right) \ .
$$

Note que todas as caixas em $L^0 = \bigcup_{k \geq p} L_k^0$ têm comprimento maior que $\frac{1}{2}$. Portanto, não podemos atravessar duas caixas de L^0 com uma reta paralela à coordenada x, no empacotamento gerado pelo Algoritmo $\mathcal{A}LS_r^*$.

Isto nos dá uma nova relação com $OPT(L)$, como segue.

Dado $L^0 = (c_1, \ldots, c_t)$, construa uma lista de retângulos $L' = (r_1, \ldots, r_t)$, onde r_j tem largura $y(c_j)$ e altura $z(c_j)$, $1 \leq j \leq t$. Usando o mesmo modelo de empacotamento usado em [2], temos que a altura do empacotamento ótimo de L^0 é igual à altura do empacotamento ótimo de L' em um retângulo de largura 1 e altura infinita. Defina a função $\overline{S}$ da área associada com a função peso como:

$$
\overline{S}(r) = z(r) W_{\mathcal{A}}(y(r)) \ .
$$

Sendo $\overline{S}(L') = \sum_{r \in L'} \overline{S}(r)$, pode-se provar que

- $\overline{S}(L') \leq U_{\mathcal{A}} OPT(L')$.

Então, temos que

$$
U_{\mathcal{A}} \cdot OPT(L) \ \geq \ U_{\mathcal{A}} \cdot OPT(L^0) = U_{\mathcal{A}} OPT(L') \geq \overline{S}(L')
$$

$$
\begin{aligned}
&= \sum_{c\in L^0} z(c)W_{\mathcal{A}}(y(c)) = \sum_{k\geq p}\sum_{c\in L_k^0} z(c)W_{\mathcal{A}}(y(c)) \\
&> \sum_{k\geq p}\left(r^{k+1}\sum_{c\in L_k^0} W_{\mathcal{A}}(y(c))\right) \geq \sum_{k\geq p} r^{k+1}(N_k^0 - C_{\mathcal{A}}) = r\mathcal{H}_1 \ .
\end{aligned}
$$

Assim,

$$
OPT(L) \geq \frac{r}{U_{\mathcal{A}}}\max\left(\mathcal{H}_1, \frac{1}{2}\mathcal{H}_1 + \frac{2}{3}\mathcal{H}_2\right) \ . \tag{3.75}
$$

Por outro lado,

$$
\begin{aligned}
\mathcal{A}LS_r^*(L) &= \sum_{k\geq p} r^k N_k \\
&= \sum_{k\geq p} r^k(N_k^0 + N_k^1 + N_k^2) \\
&= \sum_{k\geq p}\left[r^k(N_k^0 - C_{\mathcal{A}}) + r^k(N_k^1 - (1+C_{\mathcal{A}})) + r^k(N_k^2 - (1+\frac{2}{3}C_{\mathcal{A}}))\right] \\
&\quad + \sum_{k\geq p}\left(2 + \frac{8}{3}C_{\mathcal{A}}\right) r^k \\
&< \mathcal{H}_1 + \mathcal{H}_2 + \left(2 + \frac{8}{3}C_{\mathcal{A}}\right)\frac{Z}{r(1-r)} \ ,
\end{aligned}
$$

i.e.,

$$
\mathcal{A}LS_r^*(L) < \mathcal{H}_1 + \mathcal{H}_2 + \left(2 + \frac{8}{3}C_{\mathcal{A}}\right)\frac{Z}{r(1-r)} \ . \tag{3.76}
$$

Analisando a razão

$$
\alpha = \frac{\mathcal{H}_1 + \mathcal{H}_2}{\max\left(\mathcal{H}_1, \frac{1}{2}\mathcal{H}_1 + \frac{2}{3}\mathcal{H}_2\right)} \ ,
$$

concluimos que

$$
\alpha \leq 1,75 \ . \tag{3.77}
$$

Combinando as desigualdades (3.75) e (3.77), temos a desigualdade do teorema.

A justeza do limite $\frac{1,75U_{\mathcal{A}}}{r}$ é ilustrada pelo exemplo dado a seguir, devido a Li e Cheng.

Sejam N_0 e N_1 dois números grandes e $N_1 > N_0$. Sejam $L_i' = (p_1^i, p_2^i, \ldots, p_{N_i}^i)$, $i = 0, 1$, uma lista de itens tal que $OPT(L_i') = N_i' \geq N_i$ e $\mathcal{A}(L_i') \geq U_{\mathcal{A}}\, OPT(L_i') - D_{\mathcal{A}}$.

Considere as seguintes duas listas de caixas L_0 e L_1:

$$
\begin{aligned}
L_0 &= (c_1^0, c_1^0, \ldots, c_{N_0}^0) \ , \quad \text{onde} \quad c_i^0 = \left(\frac{1}{2} + \delta, p_i^0, r + \xi\right) \ , \quad 1 \leq i \leq N_0 \\
L_1 &= (c_1^1, c_1^1, \ldots, c_{N_1}^1) \ , \quad \text{onde} \quad c_i^1 = \left(\frac{2}{3}\frac{1}{2^m} + \delta, p_i^1, r + \xi\right) \ , \quad 1 \leq i \leq N_1 \ .
\end{aligned}
$$

Defina L como sendo $L = L_0||L_1$. Neste caso, temos que

$$\begin{aligned} \mathcal{A}LS_r^*(L) &\geq (U_{\mathcal{A}}N_0' - D_{\mathcal{A}})1 + \left\lceil \frac{U_{\mathcal{A}}N_1' - D_{\mathcal{A}}}{2^m} \right\rceil 1 \\ &\geq U_{\mathcal{A}}\left(N_0' + \frac{N_1'}{2^m}\right) - D_{\mathcal{A}}\left(1 + \frac{1}{2^m}\right) . \end{aligned}$$

Se forem escolhidos valores para N_0 e N_1 de forma que $\frac{N_1'}{N_2'} = \left\lfloor \frac{\left(\frac{1}{2}-\delta\right)}{\left(\frac{2}{3}2^{-m}+\delta\right)} \right\rfloor$, então $OPT(L) \leq N_0'(r+\xi)$. Esta escolha é possível para NF, BF e FF usando o Lema 2.1.3. Em [19], Lee e Lee provaram que isto também é possível para H_M. Tomando-se $\delta = 2^{-(2m+1)}$, obtém-se que

$$\frac{N_0' + \frac{N_1'}{2^m}}{OPT(L)} \geq \frac{1}{r+\xi}\left(1 + \frac{N_2'}{N_1'2^m}\right) > \frac{1}{r+\xi}\left(1 + \frac{3}{4}\frac{1-2^{-2m}}{1+3\cdot 2^{-(m+2)}} - \frac{1}{2^m}\right) .$$

Quando $m \to \infty$ e $\xi \to \infty$, a razão acima tende para $\frac{1{,}75}{r}$. Portanto, a razão $\frac{\mathcal{A}LS_r^*(L)}{OPT(L)}$ pode ficar arbitrariamente próxima de $\frac{1{,}75U_{\mathcal{A}}}{r}$, escolhendo-se N_0 suficientemente grande.

□

3.3.3 Um esquema de arredondamento melhor ainda

Usando a mesma idéia dos dois algoritmos anteriores, pode-se obter melhorias para reduzir as perdas de arredondamento na direção da largura. Vimos que o primeiro algoritmo tem um limite de desempenho esperado de $\left(\frac{1}{r}\right) \cdot 2 \cdot U_{\mathcal{A}}$, sendo o fator $\frac{1}{r}$ devido ao arredondamento da altura, o fator 2 devido ao uso da potência de $\frac{1}{2}$ no comprimento e $U_{\mathcal{A}}$ pelo uso do Algoritmo $\mathcal{A}$ na direção da largura. Vimos também que o segundo algoritmo tem limite de desempenho esperado de $\left(\frac{1}{r}\right) \cdot 1{,}75 \cdot U_{\mathcal{A}}$ é semelhante ao primeiro só que utiliza uma heurística mais sofisticada na direção do comprimento. Além disso, as faixas têm comprimentos da forma $\frac{1}{2^m}$ e $\frac{1}{3}\frac{1}{2^m}$, sendo que o valor $1{,}75$ pode ser considerado como uma média pelo uso das duas formas de comprimentos de faixa usadas.

Intuitivamente, um esquema de arredondamento mais refinado, que faz uso de uma potência de s, $0 < s < 1$, na direção do comprimento, com s próximo de 1, permitiria reduzir as perdas devido ao arredondamento. Entretanto, uma dificuldade em usar $s > \frac{1}{2}$ é que $\frac{1}{s}$ não é mais um inteiro, e portanto, uma faixa de comprimento s^m não pode ser dividida em $\frac{1}{s}$ faixas de comprimento s^{m+1}. Assim, ao invés de usar a divisão binária para criar faixas em um nível, cada nível é preenchido com faixas usando a mesma técnica dos algoritmos de empacotamento unidimensional. Em particular, Li e Cheng provaram que usando a técnica do Algoritmo FF, pode-se ter um limite de desempenho

assintótico melhor. Denominaremos este algoritmo de $\mathbf{FFLS_{r,s}}$, **First Fit Level Strip com parâmetros r, s**, $(0 < r, s < 1)$.

Da mesma forma como definido anteriormente, os níveis terão altura r^k, $k \geq 0$, sendo este chamado de **k-nível**. As faixas terão comprimento s^m, $m \geq 0$. Uma faixa de comprimento s^m em um k-nível é denominado de **(k, m)-faixa**.

Define-se **(k, m)-caixas** analogamente. Para empacotar uma (k, m)-caixa, o Algoritmo $FFLS_{r,s}$ aplica o Algoritmo FF para empacotar esta caixa em uma das (k, m)-faixas existentes. Caso não consiga, empacota esta caixa em uma nova (k, m)-faixa e aplica o Algoritmo FF novamente para empacotar a faixa em um dos k-níveis existentes.

Mais formalmente, temos:

Algoritmo $FFLS_{r,s}(L)$**,** $0 < r\,,\ s < 1$

Entrada: Lista de caixas $L = (c_1, \ldots, c_n) \in \mathcal{R}(1, 1)$.
Saída: Empacotamento $\wp$ de L para o $PET(1, 1)$

1. Para cada caixa $c \in L$, faça:

1.1. Sejam k e m inteiros tais que c seja uma (k, m)-caixa.

1.2. Seja $F_{k,m}$ o conjunto das (k, m)-faixas.

1.3. Aplique o Algoritmo FF sobre o conjunto $F_{k,m}$ para empacotar a caixa c.

1.4. Se uma nova faixa f foi criada contendo apenas a caixa c, então

1.4.1. Seja N_k o conjunto dos k-níveis.

1.4.2. Aplique o Algoritmo FF sobre N_k para empacotar na faixa f.

2. Seja $\wp$ o empacotamento construído no Passo 1.
3. Retorne $\wp$.

fim algoritmo.

O provável limite de desempenho assintótico deste algoritmo é $\frac{1}{r}\left(\frac{1,7}{s}\right)(1, 7) = \frac{2.89}{r \cdot s}$. O fator $\frac{1}{r}$ é devido ao arredondamento na altura da caixa na potência de $\frac{1}{r}$, o fator $\frac{1}{s}$ é devido ao arredondamento no comprimento em uma potência de $\frac{1}{s}$ e um dos fatores 1,7 é devido à aplicação do Algoritmo FF para empacotar a caixa na faixa, e o último fator 1,7 é devido à aplicação do Algoritmo FF para empacotar as faixas nos níveis. Assim, para $r \to 1$, $s \to 1$, temos que a razão se aproxima de 2,89. Li e Cheng mostraram que este limite é realmente válido, mas não para todos os valores de s, mas somente aqueles onde $s^t = \frac{1}{2}$ para algum $t \geq 1$, t inteiro, como provado no lema abaixo.

Lema 3.3.3 *Se $s^t \neq \frac{1}{2}$ para qualquer inteiro $t > 0$, então há uma instância L para o Algoritmo $FFLS_{r,s}$ tal que a razão $\frac{FFLS_{r,s}(L)}{OPT(L)}$ pode chegar tão próxima de $\frac{3,4}{r}$ quanto se queira.*

Dem. Assuma que $s^{t+1} < \frac{1}{2} < s^t$. Seja $L' = (p_1, p_2, \ldots, p_n)$ uma lista de itens tal que $OPT^b(L') = K$ e $FF(L') > 1,7K - 8$, onde K é um inteiro par grande. Considere um lista de caixas $L = (c_1, c_2, \ldots, c_n)$, construída a partir de L', tal que $c_i = (r + \xi, s^{t+1} + \xi, p_i)$, $1 \leq i \leq n$. Claramente, no empacotamento gerado pelo Algoritmo $FFLS_{r,s}$, cada nível tem altura 1, contendo apenas uma faixa de comprimento s^t. Portanto, $FFLS_{r,s}(L) \geq 1,7K - 8$. Por outro lado, se escolhermos ξ bem pequeno, tal que $s^{t+1} + \xi \leq \frac{1}{2}$, então pode-se ter duas faixas por nível, e neste caso, $OPT(L) \leq \left(\frac{K}{2}\right)(r + \xi)$. Quando $K \to \infty$ e $\epsilon \to 0$, a razão $\frac{FFLS_{r,s}(L)}{OPT(L)}$ pode se aproximar de $\frac{3,4}{r}$ tanto quanto se queira.

□

Os dois lemas seguintes, provados em [24], são usados na demonstração do limite de desempenho assintótico do algoritmo $FFLS_{r,s}$ quando $s^t = \frac{1}{2}$ para algum $t > 0$.

Lema 3.3.4 *Seja $L = (c_1, c_2, \ldots, c_n)$ uma lista de caixas. Seja $L' = (c'_1, c'_2, \ldots, c'_n)$ outra lista de caixas, onde $z(c'_i) = z(c_i)$, $x(c_i)$ e $y(c'_i) = W_{FF}(y(c_i))$, $1 \leq i \leq n$. Se L pode ser empacotada em uma caixa B de altura h, comprimento k, largura 1, então L' pode ser empacotada em uma caixa B' de altura h, comprimento k e largura 1,7.*

Para $0 < s < 1$, defina $f_s(x) = s^{\lfloor \log_s(x) \rfloor}$, $0 < x < 1$, i.e., se $s^{m+1} < x < s^m$, então $f_s(x) = s^m$. Para $0 \leq a \leq 1$, defina $W_{FF,s}(a) = W_{FF}(f_s(a))$. Seja $\mathcal{I} = \{I = (p_1, \ldots, p_n) : p_i \in [0,1]$ e $\sum_{i=1}^n p_i \leq 1\}$ e $\lambda(s) = \sup_{I \in \mathcal{I}}(W_{FF,s}(I))$ e $\psi(s) = \sum_{m \geq 0} W_{FF}(s^m)$.

Lema 3.3.5 *Seja $L = (c_1, \ldots, c_n)$ uma lista de caixas. Seja $L' = (c'_1, c'_2, \ldots, c'_n)$ outra lista de caixas, onde $z(c'_i) = z(c_i)$, $x(c'_i) = W_{FF,s}(x(c_i))$ e $y(c'_i) = y(c_i)$, $1 \leq i \leq n$. Se L pode ser empacotada em uma caixa B de altura h, comprimento 1 e largura w, então L' pode ser empacotada em uma caixa B' de altura h, comprimento $\lambda(s)$ e largura w.*

Teorema 3.3.6 *Para quaisquer constantes r e s, $0 < r$, $s < 1$, $s^t = \frac{1}{2}$ para algum $t \geq 1$, e qualquer lista de caixas L na qual nenhuma caixa tem altura superior a Z, temos que*

$$FFLS_{r,s}(L) < \frac{1,7\lambda(s)}{r} OPT(L) + \left(\frac{2 + 2\psi(s)}{r(1-r)}\right) Z \ ,$$

onde $\lim_{t \to \infty} \lambda(s) = 1,7$ e $\psi(s) = O((1-2)^{-1})$.

Dem. Defina uma nova função $\overline{V}$, correspondente ao volume associado à função peso, da seguinte forma:

$$\overline{V}_s(c) = z(c)W_{FF,s}(x(c))W_{FF}(y(c)) \ .$$

Vamos provar que

$$\overline{V}_s(L) \leq 1,7\lambda(s)OPT(L) \ .$$

Suponha que L possa ser empacotada em uma caixa de altura h, comprimento 1 e largura 1. Seja $L' = (c'_1, c'_2, \ldots c'_n)$ uma lista de caixas, onde $z(c'_i) = z(c_i)$, $x(c'_i) = x(c_i)$ e $y(c'_i) = W_{FF}(y(c_i))$, $1 \leq i \leq n$. Pelo Lema 3.3.4, temos que L' pode ser empacotada em uma caixa B' de altura h, comprimento 1 e largura 1,7. Seja $L'' = (c''_1, c''_2, \ldots, c''_n)$ outra lista de caixas onde $z(c''_i) = z(c'_i)$, $x(c''_i) = W_{FF,s}(x(c'_i))$ e $y(c''_i) = y(c'_i)$, $1 \leq i \leq n$. Pelo Lema 3.3.5, temos que L'' pode ser empacotada em uma caixa B'' de altura h, comprimento $\lambda(s)$ e largura 1,7. Assim, $\overline{V}_s(L) \leq 1,7\lambda(s)h$. Colocando $h = OPT(L)$, o fato segue.

Seja $\overline{V}_{k,s} = \overline{V}_s(L_k)$. Observe que

$$\begin{aligned}
\overline{V}_{k,s} &= \sum_{c \in L_k} z(c)W_{FF,s}(x(c))W_{FF}(y(c)) \\
&> r^{k+1} \sum_{m \geq 0} \left(W_{FF}(s^m) \sum_{c \in L_{k,m}} W_{FF}(y(c)) \right) \\
&\geq r^{k+1} \sum_{m \geq 0} W_{FF}(s^m)(n_{k,m} - 2) \\
&= r^{k+1} \left(\sum_{m \geq 0} W_{FF}(s^m) n_{k,m} - 2 \sum_{m \geq 0} W_{FF}(s^m) \right) \ .
\end{aligned}$$

Seja S_k o conjunto das faixas não vazias em todos os k-níveis. Como as faixas de S_k também são empacotadas nos k-níveis usando o Algoritmo FF, temos que $W_{FF}(S_k) \geq N_k - 2$. Como $W_{FF}(S_k) = \sum_{m \geq 0} W_{FF}(s^m) n_{k,m}$, obtemos que

$$\begin{aligned}
\overline{V}_s(L) &= \sum_{k \geq p} \overline{V}_{k,s} \\
&> r \sum_{k \geq p} r^k (N_k - 2 - 2\psi(s)) \\
&= r \left(FFLS_{r,s}(L) - (2 + 2\psi(s)) \frac{r^p}{1 - r} \right) \ .
\end{aligned}$$

Pelo fato anteriormente provado, temos que $1,7\lambda(s)OPT(L) \geq \overline{V}_s(L)$. Esta desigualdade juntamente com a desigualdade acima nos fornece desigualdade desejada. Por fim, falta provar que $\lim_{t \to \infty} \lambda(s) = 1,7$ e $\psi(s) = O((1-s)^{-1})$. Não reproduziremos aqui a demonstração desses resultados que pode ser encontrada em [24]. □

Vale aqui chamar a atenção do leitor, quanto ao fato de que os algoritmos apresentados nesta seção diferem substancialmente dos algoritmos vistos na seção anterior. Os algoritmos desta seção são baseados numa primeira partição da instância em função da altura das caixas, enquanto os algoritmos da seção 3.2 particionam a instância quanto ao fundo das caixas.

3.4 Comentários

Neste capítulo, vimos vários algoritmos para o PET. O algoritmo com melhor limite de desempenho assintótico é o Algoritmo $FFLS_{r,s}$, $0 < r, s < 1$. Vimos que $FFLS_{r,s}(L) \leq \alpha \cdot OPT(L) + \beta \cdot Z$, onde α pode chegar tão próximo de $2,89$ quanto se queira, sendo que para isto basta colocar $r, s \to 1$. Infelizmente, a constante aditiva β do limite de desempenho assintótico é demasiadamente grande, como referido por seus autores em [25]. Isto ocorre pois a constante aditiva inclui o produto de dois números grandes $\left(\frac{1}{r(1-r)} \cdot \frac{1}{s(1-s)}\right)$. A razão disto é que à medida que r e s se aproximam de 1, as caixas tendem a ser empacotadas em diferentes níveis e diferentes faixas. Isto faz com que aumente o número de faixas e níveis incompletos.

Outro algoritmo com limite de desempenho assintótico um pouco maior, é o Algoritmo $\mathcal{A}LS_r^*$, $\mathcal{A} \in \{NF, FF, BF, H_M\}$. Calculando a constante aditiva β para quando $\mathcal{A} = FF$, e fixando valores para α, obtemos os seguintes resultados:

$$\begin{array}{lll} \alpha = 3,25 & \text{e} & \beta = 94, \\ \alpha = 3,04904 & \text{e} & \beta = 309, \\ \alpha = 3 & \text{e} & \beta = 887. \end{array}$$

Como podemos ver, quando $\alpha = 3$ a constante β possui um valor relativamente grande.

Cabe aqui notar que, no caso do Algoritmo T, apresentado na Seção 3.2.11, vimos que

$$T(L) \leq 3 \cdot OPT(L) + \frac{141}{8} Z.$$

Ou seja, o limite de desempenho assintótico do Algoritmo T é menor ou igual a 3 e a constante aditiva β é menor que 18.

Estas observações devem ser levadas em conta ao refletirmos sobre as vantagens e as desvantagens de cada algoritmo. Note ainda que, além da garantia de desempenho, há que se considerar ainda os aspectos relativos à sua implementação. Não discutiremos aqui estes aspectos, já que a descrição de cada algoritmo permite concluir quando a implementação é mais simples ou mais trabalhosa.

Capítulo 4

Algoritmos para o Problema do Empacotamento Tridimensional Ortogonal Orientado na Dimensão z

Neste capítulo trataremos do Problema do Empacotamento Tridimensional Ortogonal e Orientado na Dimensão z, PET^R. Este problema foi estudado por Li e Cheng [23], que desenvolveram algoritmos usando técnicas semelhantes às do algoritmo G_1, descrito no Capítulo 3. Os limites de desempenho assintótico desses algoritmos não se mostraram, porém, melhores que os do Algoritmo G_1. Lembramos aqui que o limite de desempenho assintótico do Algoritmo G_1 é $4\frac{4}{7}$.

Apresentaremos aqui três algoritmos que desenvolvemos. Um deles é para o PET^R e tem um limite de desempenho assintótico menor que $3,04904$. Os dois outros são para o PET^R restrito a instâncias especiais. Nesses dois casos exibiremos algoritmos com limites de desempenho assintótico melhores. Esses resultados não foram encontrados na literatura.

No Apêndice C, mencionamos uma aplicação interessante do PET^R.

4.1 Definição do PET^R

O **Problema do Empacotamento Tridimensional Ortogonal e Orientado na Dimensão z**, denotado por $\mathbf{PET^R}$, é semelhante ao PET.

A principal diferença entre ambos, é que no PET não é permitido fazer rotações das caixas, enquanto que no PET^R é permitido fazer um tipo de rotação. Mais precisamente, no PET^R as caixas a serem empacotadas podem sofrer rotações ortogonais sobre o fundo (veja Figura 4.1). Veremos a seguir uma definição mais formal do PET^R.

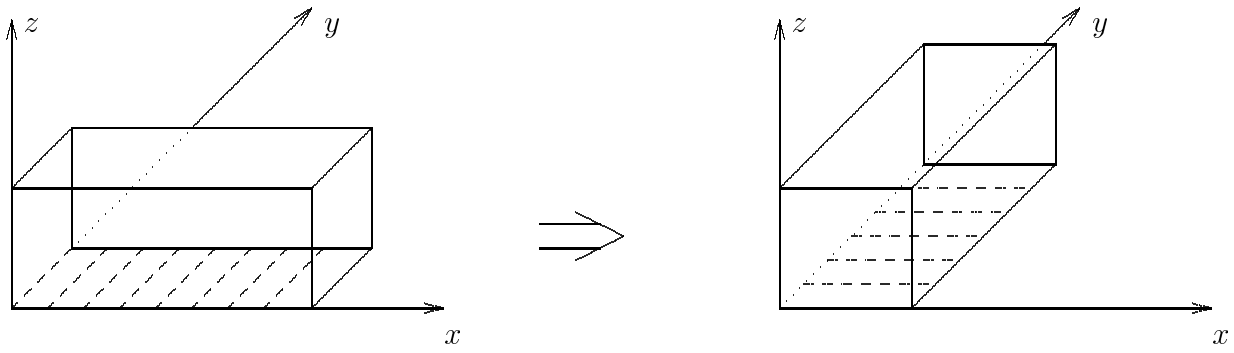


Figura 4.1: Rotação ortogonal do fundo, orientado na dimensão z

Seja $L = (c_1, c_2, \ldots, c_n)$ uma lista de caixas, onde $c_i = (x_i, y_i, z_i)$. Para cada caixa c_i definimos $\rho(\mathbf{c_i})$ como sendo a caixa $\rho(c_i) = (y_i, x_i, z_i)$. Dada uma lista L, seja $\mathbf{\Gamma(L)} = \{(d_1, d_2, \ldots, d_n) : d_i \in \{c_i, \rho(c_i)\}\}$.

O **Problema do Empacotamento Tridimensional Ortogonal e Orientado na Dimensão z**, $\mathbf{PET^R}$, pode ser assim definido:

Dados uma caixa $B = (a, b, \infty)$ e uma lista de caixas $L = (c_1, c_2, \ldots, c_n)$, encontrar uma lista $L' \in \Gamma(L)$ e um empacotamento $\wp$ de L' em B tal que $H(\wp)$ é mínimo. Ou seja, $H(\wp) = \min\{H(\wp') : \ \wp'$ é um empacotamento de L' em $B,\ L' \in \Gamma(L)\}$.

Neste problema, cada caixa em $L = (c_1, \ldots, c_n)$ pode ser rotacionada ou não. Para indicar como cada caixa $c_i \in L$ foi empacotada por um dado empacotamento $\wp$ de uma lista $(d_1, \ldots, d_n) \in \Gamma(L)$, usaremos o vetor $\wp^\rho \in \{0, 1\}^n$, onde $\wp^\rho(d_i) = 1$ se $d_i = c_i$, e $\wp^\rho(d_i) = 0$ se $d_i = \rho(c_i)$.

Como no caso do PET, a notação $\mathbf{PET^R(a, b)}$ será usada para denotarmos o PET^R cuja caixa B é da forma (a, b, ∞).

Analogamente, se L é uma instância do PET^R, então denotaremos por $\mathbf{OPT^R(L)}$ a altura do empacotamento ótimo da lista L.

Denotaremos também por $\mathcal{R}^\mathbf{R}(\mathbf{a}, \mathbf{b})$ o conjunto de instâncias possíveis para o $PET^R(a, b)$. Este conjunto contém listas cujas caixas $c_i = (x_i, y_i, z_i)$ são tais que $x_i \leq a$ e $y_i \leq b$ ou $x_i \leq b$ e $y_i \leq a$.

4.2 $PET \times PET^R$: resultados em comum

Nesta seção, mostraremos que o PET pode ser reduzido ao PET^R. Além disso, veremos que se existe um algoritmo $\mathcal{A}^R$ para o PET^R com um limite de desempenho assintótico α, então existe um algoritmo $\mathcal{A}$ para o PET com um limite de desempenho assintótico α.

Teorema 4.2.1 *O PET pode ser polinomialmente reduzido ao PET^R. Mais ainda, se $\mathcal{A}^R$ é um algoritmo para o PET^R, tal que*

$$\mathcal{A}^R(L) \leq \alpha \cdot OPT^R(L) + \beta \cdot Z$$

para toda lista L onde nenhuma caixa tem altura superior a Z, então existe um algoritmo $\mathcal{A}$ para o PET, tal que

$$\mathcal{A}(L) \leq \alpha \cdot OPT(L) + \beta \cdot Z,$$

para toda lista L onde nenhuma caixa tem altura superior a Z.

Dem. Seja L uma instância do $PET(a,b)$. Considere o seguinte algoritmo $\mathcal{A}$. Este algoritmo faz primeiramente a reparametrização de B para $B' = (a', b, \infty)$ e L para L', na mesma proporção, de forma que $\min\{x(c) : c \in L'\} > b$. A seguir, comporta-se como o algoritmo $\mathcal{A}^R$, recebendo (L', a', b) como entrada. Finalmente, após obter um empacotamento $\wp'$ de L' em $B' = (a', b, \infty)$, retorna $\wp'$ à parametrização original, obtendo $\wp$. Claramente, $\mathcal{A}(L) \leq \alpha \cdot OPT(L) + \beta Z$, se α é o limite de desempenho assintótico do algoritmo $\mathcal{A}^R$.

□

Uma primeira idéia para resolver o PET^R é tentar aproveitar os algoritmos do PET. Porém surge um problema quanto à orientação das caixas, pois os algoritmos do PET já supõem que todas as caixas têm uma orientação fixa segundo a qual todas as caixas podem ser empacotadas nesta orientação inicial.

Uma redução natural que surge é construir para cada instância $L = (c_1, c_2, \ldots, c_n) \in \mathcal{R}^R(a,b)$ uma nova instância $\phi(L) \in \mathcal{R}(a,b)$ tal que

$$\phi(L) = (d_1, d_2, \ldots, d_n) \ ,$$

onde $d_i = \begin{cases} c_i & \text{se } \ x_i \leq a \ \text{ e } \ y_i \leq b, \\ \rho(c_i) & \text{caso contrário .} \end{cases}$

Feito isto, aplicar um algoritmo do PET sobre $\phi(L)$.

Assim, para cada algoritmo $\mathcal{A}$ do PET, definimos $\widehat{\mathcal{A}}$ como sendo o seguinte algoritmo para o PET^R: para toda instância L do $PET^R(a,b)$, $\widehat{\mathcal{A}}$ aplica o algoritmo $\mathcal{A}$ sobre a lista $\phi(L)$.

Apesar da construção acima gerar algoritmos perfeitamente viáveis para o PET^R, esta transformação não garante que os algoritmos assim concebidos continuem tendo o mesmo limite de desempenho assintótico do algoritmo original.

No lema abaixo, mostramos que nenhum algoritmo $\widehat{\mathcal{A}}$ para o PET^R, construído a partir de um algoritmo $\mathcal{A}$ do PET, conforme descrevemos, tem limite de desempenho assintótico menor que 3.

Lema 4.2.2 *Se $\widehat{\mathcal{A}}$ é um algoritmo para o PET^R construído a partir de um algoritmo $\mathcal{A}$ do PET, conforme a construção acima, então*

$$r(\widehat{\mathcal{A}}) \geq 3.$$

Dem. Para ver isto, considere a seguinte instância $L = (c_1, c_2, \ldots, c_{3k})$ do $PET^R(4 - 2\epsilon, 2)$, onde $c_1 = c_2 = \ldots = c_{3k} = (2, 1 + \epsilon, 1)$ e k é um inteiro positivo.

Primeiramente, observe que é possível empacotar L em k níveis, i.e., $\mathbf{OPT^R(L) = k}$. Para verificar isto, rotacione cada caixa de L previamente e construa um empacotamento colocando três caixas por nível. Note também que $L = \phi(L)$.

Por outro lado, qualquer algoritmo $\mathcal{A}$ do PET é tal que $\widehat{\mathcal{A}}(L) \geq 3k$, já que qualquer algoritmo do PET empacota cada caixa de L em apenas um nível. Assim,

$$r(\mathcal{A}) \geq \lim_{k\to\infty} \sup_{L} \frac{\mathcal{A}(L)}{OPT(L)} \geq 3.$$

Isto finaliza a demonstração do lema.

□

4.3 Algoritmo R^R: modificação do Algoritmo R

Nesta seção mostraremos como o Algoritmo R desenvolvido para o PET (veja a Seção 3.2.10) pode ser modificado de modo que sirva para o PET^R, e continue apresentando o mesmo limite de desempenho assintótico.

Primeiramente observamos que o Lema 3.2.1 é válido para o PET^R, i.e., $OPT^R(L) \geq \frac{V(L)}{ab}$. Assim, todas as operações envolvendo o volume das caixas se mantêm válidas no caso do PET^R. Note porém que as operações que fazem uso das desigualdades dadas pelo Lema 3.2.14 não são válidas para o PET^R. Observe que se $L_A = (c_1, c_2, \ldots, c_n)$ é uma sublista de L onde $x_i > \frac{a}{2}$ e $y_i > \frac{b}{2}$ $(i = 1, 2, \ldots, n)$ então no PET duas caixas de L_A não podem ser empacotadas em um mesmo nível, enquanto no PET^R pode haver casos em que se pode empacotar duas caixas de L_A no mesmo nível. Portanto, antes de executar o

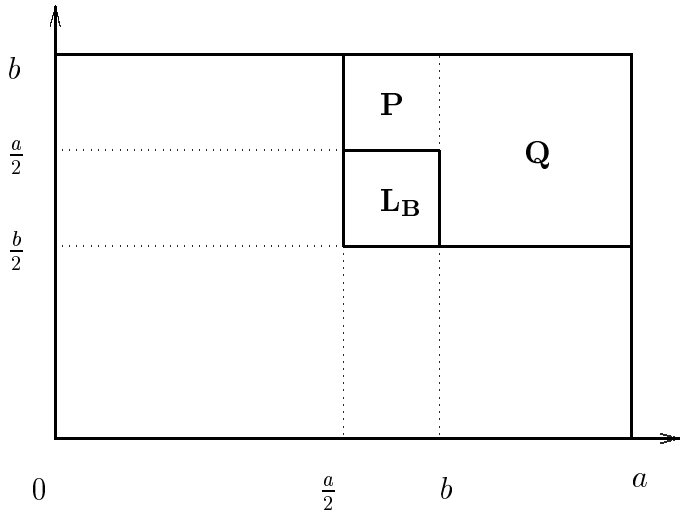


Figura 4.2: Sublistas de L_A.

Algoritmo R, devemos remover todas as caixas de L_A que podem ser empacotadas duas a duas em um mesmo nível. Feito isto, o Lema 3.2.14 é válido para as caixas restantes. Assim, a demonstração do Teorema 3.2.25 poderá ser aproveitada desde que consigamos provar a validade do Lema 3.2.14 para as caixas em L_A.

Vamos supor que a caixa $B = (a, b, \infty)$ é tal que $a \geq b$ (para $a < b$ o caso é análogo).

Seja $L_A \subseteq L$ tal que $L_A = \{c \in L : x(c) > \frac{a}{2} \text{ e } y(c) > \frac{b}{2}\}$ e seja $L_B \subseteq L_A$ tal que $L_B = \{c \in L_A : x(c) \leq b \text{ e } y(c) \leq \frac{a}{2}\}$. Suponha que $a \leq 2b$, pois caso contrário, L_B é vazio, e o Lema 3.2.14 é válido para L_A, como veremos adiante.

Seja $L_B = (c_1, c_2, \ldots, c_{n_B})$. Rotacione as caixas de L_B, gerando outra lista $L'_B = (c'_1, c'_2, \ldots, c'_{n_B})$ onde $c'_i = \rho(c_i)$, $i = 1, 2, \ldots, n_B$. Aplique o Algoritmo $NFDH^x$ a L'_B e gere um empacotamento $\wp_B$. Feito isto, aplique o Algoritmo R a $L \setminus L_B$, obtenha um empacotamento desta lista e concatene-o com $\wp_B$, obtendo assim um empacotamento de L.

Note que $L'_B \subseteq \mathcal{C}\left[\frac{b}{2a}, \frac{1}{2} \ ; \ \frac{a}{2b}, 1\right](a, b)$, e portanto pelo Lema 3.2.10,

$$H(\wp_B) \leq 2\frac{V(L_B)}{ab} + Z. \tag{4.1}$$

Observe também que não podemos colocar duas caixas de $L_A \setminus L_B$ em um mesmo nível. Para verificar isto, note que a lista $L_A \setminus L_B$ pode ser particionada em duas sublistas, P e Q, onde $P = \{c \in L_A \setminus L_B : x(c) \leq b\}$ e $Q = \{c \in L_A \setminus L_B : x(c) > b\}$ (veja a Figura 4.2). As caixas de Q não podem ser rotacionadas, e portanto não é possível colocar duas caixas de Q em um mesmo nível. Duas caixas de P também não podem estar em um mesmo nível, pois tanto $x(c)$ como $x(\rho(c))$ são maiores que $\frac{a}{2}$, para toda caixa c de P. Finalmente, não pode haver uma caixa de P e outra de Q em um mesmo nível, pois dado $c_P \in P$ e $c_Q \in Q$, tanto $y(c_P) + y(c_Q) > b$ como $y(\rho(c_P)) + y(c_Q) > b$ e tanto $x(c_P) + x(c_Q) > a$ como $x(\rho(c_P)) + x(c_Q) > a$. Logo, o Lema 3.2.14 é válido para um empacotamento $\wp_1$ gerado pelo algoritmo UC quando aplicado a $L_A \setminus L_B$.

Pelo acima exposto fica justificado o Algoritmo R^R, descrito a seguir.

Algoritmo $R^R(L)$

Entrada: Constantes a e b e lista de caixas $L \in \mathcal{R}^R(a,b)$.
Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. $L \leftarrow \phi(L)$.
2. Se $a \geq b$ então

2.1. Construa a sublista L_A, tomando

$$L_A = \{c \in L : x(c) > \frac{a}{2} \text{ e } y(c) > \frac{b}{2}\} .$$

2.2. Construa a sublista $L_B \subseteq L_A$, tomando

$$L_B = \{c \in L_A : x(c) \leq b \text{ e } y(c) \leq \frac{a}{2}\} .$$

2.3. Suponha que $L_B = (c_1, c_2, \ldots, c_{n_B})$, $L_B \neq \emptyset$. Seja $L'_B = (\rho(c_1), \rho(c_2), \ldots, \rho(c_n))$. $\wp_B \leftarrow NFDH^x(L'_B)$.

2.4. $\wp' \leftarrow R(L \setminus L_B)$.

2.5. Se $L_B \neq \emptyset$ então $\wp \leftarrow \wp_B \| \wp'$, senão $\wp \leftarrow \wp'$.

3. Se $a < b$ então construa um empacotamento $\wp$ análogo ao definido no Passo 1 (permutando o papel de x e y).
4. Retorne $\wp$.

fim algoritmo.

Teorema 4.3.1 *Para qualquer instância L do PET^R tal que nenhuma caixa tem altura maior que Z, tem-se que*

$$H(\wp) < 3,04904 \cdot OPT^R(L) + 15Z.$$

Dem. Segue analogamente à demonstração do Teorema 3.2.25, usando adicionalmente a desigualdade (4.1).

□

No caso de outros algoritmos vistos no Capítulo 3, se as demonstrações dos limites de desempenho usam o mesmo esquema de comparação com volume e comparação da altura ótima (como feito para a lista L_A) então fazendo uma modificação na prova análoga à que fizemos para o Algoritmo R, obtemos resultados análogos.

4.4 Algoritmo P^R: para caixas pequenas

Em muitos problemas, o fundo da caixa B tem dimensões bem maiores que as caixas a serem empacotadas, permitindo que todas as caixas possam ser rotacionadas. Mostraremos nesta seção que é possível obter um algoritmo com limite de desempenho melhor para este caso. Assumiremos aqui que a lista de entrada $L = (c_1, \ldots, c_n)$ para o PET^R satisfaz a condição

$$\max_{i=1,\ldots,n} \{x_i, y_i\} \leq \min\{a, b\} \ .$$

O algoritmo que apresentaremos nesta seção usa as mesmas estratégias usadas no algoritmo R do capítulo 3. Este algoritmo combina duas sublistas da instância inicial, que isoladamente garantem pouca área por nível (em relação ao Lema 3.2.2), combinando-as é possível eliminar uma delas e garantir um empacotamento final melhor.

Vimos que no Algoritmo C as sublistas que constituem o *gargalo* do empacotamento são aquelas que garantem $\frac{ab}{3}$ e $\frac{ab}{4}$ de área por nível. Vimos também que algumas das listas que garantem $\frac{ab}{3}$ de área podem ser redivididas garantindo dessa forma uma área melhor (isto foi feito no Algoritmo R). Como $\max\{x_i, y_i\}_{i=1,\ldots,n} \leq \min\{a, b\}$, podemos rotacionar as caixas de forma a ficar com apenas duas sublistas com pouca garantia de área. Feito isto, combinamos estas duas listas de modo que todas as caixas de uma delas sejam consumidas no empacotamento combinado.

Algoritmo $\mathbf{P^R(L)}$

Entrada: Constantes a e b e lista de caixas $L = (c_1, \ldots, c_n)$ para o $PET^R(a,b)$ que satisfaz $\max_{i=1,\ldots,n}\{x_i, y_i\} \leq \min\{a,b\}$.

Saída: Empacotamento $\wp$ de L em $B = (a, b, \infty)$.

1. Se $a \geq b$ então

1.1. Rotacione as caixas de L e obtenha uma nova lista com caixas c tais que $x(c) \leq y(c)$. Chame de L a nova lista.

1.2. Divida a lista L em sublistas $L'_1, L''_1, L'_2, L''_2, L_3, \ldots, L_8$ como segue (veja a Figura 4.3).

$$L'_1 = L \bigcap \mathcal{C}\left[\tfrac{1}{2}, \tfrac{5}{8}\ ;\ \tfrac{1}{2}, 1\right](a,b), \quad L''_1 = L \bigcap \mathcal{C}\left[\tfrac{5}{8}, 1\ ;\ \tfrac{5}{8}, 1\right](a,b),$$
$$L'_2 = L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{3}{8}\ ;\ \tfrac{1}{2}, 1\right](a,b), \quad L''_2 = L \bigcap \mathcal{C}\left[\tfrac{3}{8}, \tfrac{1}{2}\ ;\ \tfrac{1}{2}, 1\right](a,b),$$
$$L_3 = L \bigcap \mathcal{C}\left[\tfrac{1}{4}, \tfrac{1}{3}\ ;\ \tfrac{1}{2}, 1\right](a,b), \quad L_4 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ \tfrac{1}{2}, 1\right](a,b),$$
$$L_5 = L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{1}{2}\ ;\ \tfrac{1}{3}, \tfrac{1}{2}\right](a,b), \quad L_6 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{3}\ ;\ \tfrac{1}{3}, \tfrac{1}{2}\right](a,b),$$
$$L_7 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{3}\ ;\ \tfrac{1}{4}, \tfrac{1}{3}\right](a,b), \quad L_8 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ 0, \tfrac{1}{4}\right](a,b).$$

1.3. $(\wp_{1,2}, R_{1,2}) \leftarrow DC^x(L'_2, L'_1)$ Seja $L_{1,2}$ as caixas de $\wp_{1,2}$.

1.4. $L_1 \leftarrow (L'_1 \cup L''_1) \setminus L_{1,2}$.

1.5. $L_2 \leftarrow (L'_2 \cup L''_2) \setminus L_{1,2}$.

1.6. $\wp_1 \leftarrow UC(L_1)$.

1.7. $\wp_i \leftarrow NFDH^x(L_i),\ i = 2, \ldots, 7$.

1.8. $\wp_8 \leftarrow G_4(L_8)$.

1.9. $\wp \leftarrow \wp_{1,4} \| \wp_1 \| \ldots \| \wp_8$.

2. Se $a < b$ então construa um empacotamento $\wp$ análogo ao definido no Passo 1 (permutando o papel de x e y).

3. Retorne $\wp$.

fim algoritmo.

Teorema 4.4.1 *Se $L = (c_1, \ldots, c_n)$ é uma instância do $PET^R(a,b)$, tal que* $\min\{a,b\} \geq \max_{i=1,\ldots,n}\{x_i, y_i\}$ *então*

$$P^R(L) \leq 3 \cdot OPT^R(L) + 9Z\ ,$$

onde $Z = \max\{z_i :\ z_i \in L\}$.

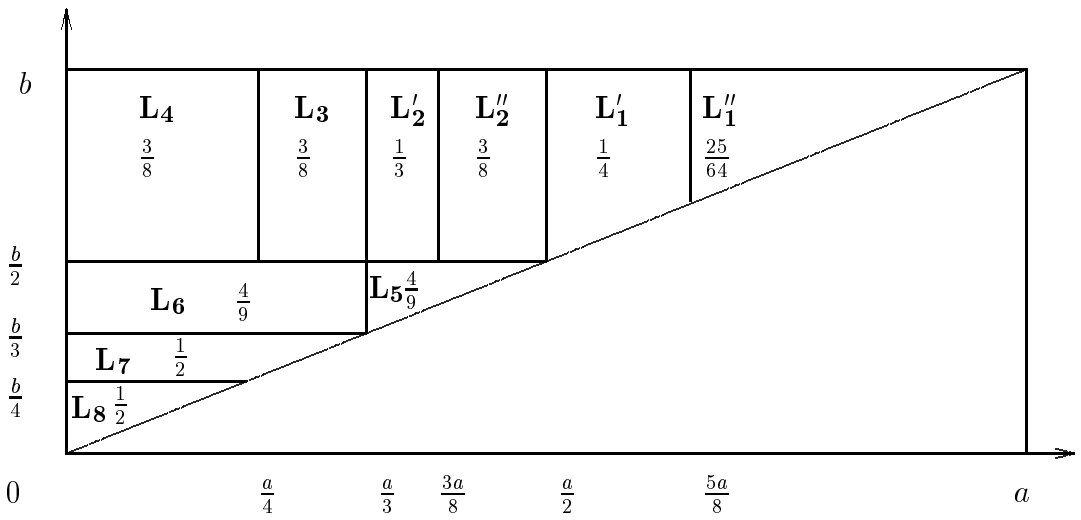


Figura 4.3: Partição da lista L efetuada pelo Algoritmo P^R.

Dem. Note que não existe lista $L_B \subseteq L$ tal que $L_B = \{c_i \in L : \frac{a}{2} < x_i \leq b \text{ e } \frac{b}{2} < y_i \leq \frac{a}{2}\}$. Portanto, o Lema 3.2.14 é aplicável à lista L_1.

O empacotamento $\wp_{1,2}$, gerado pelo algoritmo DC^x aplicado às listas L'_1 e L'_2, faz com que uma destas duas listas seja totalmente consumida no empacotamento $\wp_{1,2}$.

Como $S(c) \geq \frac{ab}{4}$ para toda caixa $c \in L'_1$ e $S(c) \geq \frac{ab}{6}$ para toda caixa $c \in L'_2$, temos pelo Lema 3.2.24 que

$$\begin{aligned} H(\wp_{1,2}) &\leq \frac{12}{5}\frac{V(L_{1,2})}{ab} + Z \\ &\leq \frac{8}{3}\frac{V(L_{1,2})}{ab} + Z. \end{aligned} \tag{4.2}$$

Note que as listas $L_3, \ldots, L_7$ são empacotadas pelo algoritmo $NFDH^x$, pelos Lemas 3.2.10 e 3.2.11 temos que

$$H(\wp_3) \leq \frac{8}{3}\frac{V(L_3)}{ab} + Z, \tag{4.3}$$

$$H(\wp_4) \leq \frac{8}{3}\frac{V(L_4)}{ab} + Z, \tag{4.4}$$

$$\begin{aligned} H(\wp_5) &\leq \frac{9}{4}\frac{V(L_5)}{ab} + Z \\ &\leq \frac{8}{3}\frac{V(L_5)}{ab} + Z, \end{aligned} \tag{4.5}$$

$$\begin{aligned} H(\wp_6) &\leq \frac{9}{4}\frac{V(L_6)}{ab} + Z \\ &\leq \frac{8}{3}\frac{V(L_6)}{ab} + Z, \qquad (4.6) \\ H(\wp_7) &\leq 2\frac{V(L_6)}{ab} + Z \\ &\leq \frac{8}{3}\frac{V(L_6)}{ab} + Z. \qquad (4.7) \end{aligned}$$

Já o empacotamento $\wp_8$ é gerado pelo Algoritmo G_4. Como $L_8 \in R_4(a,b)$, pelo Lema 3.2.6,

$$\begin{aligned} H(\wp_8) &\leq 2\frac{V(L_7)}{ab} + Z \\ &\leq \frac{8}{3}\frac{V(L_7)}{ab} + Z. \qquad (4.8) \end{aligned}$$

Temos dois casos a considerar conforme o resultado da combinação das listas L_1' e L_2'.

Caso 1: $\mathbf{L_2' \subseteq L_{1,2}}$

Neste caso, todas as caixas de L_2' foram consumidas no empacotamento $\wp_{1,2}$. Logo, $L_2 \subseteq \mathcal{C}\left[\frac{3}{8}, \frac{1}{2}\ ;\ \frac{1}{2}, 1\right](a,b)$ e portanto,

$$H(\wp_2) \leq \frac{8}{3}\frac{V(L_2)}{ab} + Z. \qquad (4.9)$$

Juntando as desigualdades (4.2) a (4.9) temos que

$$H(\wp_i) \leq \frac{8}{3}\frac{V(L_i)}{ab} + Z, \quad i \in \{(1,2), 2, \ldots, 8\}. \qquad (4.10)$$

Por outro lado, como $S(c) > \frac{ab}{4}$, $\forall c \in L_1$, temos pelo Lema 3.2.13

$$H(\wp_1) \leq 4\frac{V(L_1)}{ab}. \qquad (4.11)$$

Como o Lema 3.2.14 pode ser aplicado à lista L_1, temos que $OPT^R(L) \geq H(\wp_1)$. Prosseguindo analogamente como foi feito na demonstração do Teorema 3.2.25, temos que

$$H(\wp) \leq \alpha_1 \cdot OPT^R(L) + 8Z,$$

onde $\alpha_1 = \frac{h+H}{max\{h, \frac{h}{4}+\frac{3}{8}H\}}$. Calculando α_1, temos que $\alpha_1 \leq 3$.

Caso 2: $\mathbf{L'_1 \subseteq L_{1,2}}$

Neste caso temos que $L_2 \subseteq \mathcal{C}\left[\frac{1}{3}, \frac{1}{2}\ ;\ \frac{1}{2}, 1\right](a,b)$ e portanto,

$$H(\wp_2) \leq 3\frac{V(L_2)}{ab} + Z. \tag{4.12}$$

Juntando as desigualdades (4.2) a (4.8) e (4.12) temos

$$H(\wp_i) \leq 3\frac{V(L_i)}{ab} + Z, \quad i \in \{(1,2), 2, \ldots, 8\}. \tag{4.13}$$

Como $L_1 = L''_1$, temos que $S(c) \geq \frac{25}{64}ab,\ \forall c \in L_1$. Pelo Lema 3.2.13, temos que

$$H(\wp_1) \leq \frac{64}{25}\frac{V(L_1)}{ab}. \tag{4.14}$$

Assim, da mesma forma que no Caso 1,

$$H(\wp) \leq \alpha_2 \cdot OPT^R(L) + 8Z,$$

onde $\alpha_2 = \frac{h+H}{max\{h, \frac{25}{64}h + \frac{1}{3}H\}}$. Calculando α_2, temos que $\alpha_2 \leq 3$.

Dos casos 1 e 2, o teorema segue. □

4.5 Algoritmo Q^R: para empacotar em caixa de fundo quadrado

No Capítulo 3 vimos o Algoritmo CQQ para o PET que empacota uma lista de caixas de fundo quadrado em uma caixa B também com fundo quadrado. Este algoritmo possui um limite de desempenho assintótico igual a 2,6875.

Apresentamos nesta seção um algoritmo para o PET^R que desenvolvemos, chamado Algoritmo Q^R, que generaliza o resultado do algoritmo CQQ. O Algoritmo Q^R empacota uma lista de caixas L qualquer em uma caixa B de fundo quadrado e tem um limite de desempenho assintótico 2,6875. Este algoritmo usa a mesma idéia do Algoritmo C do Capítulo 3, i.e., divide a lista L em sublistas e empacota cada uma delas com um algoritmo específico.

Antes de apresentar o algoritmo Q^R, definiremos outra notação que será conveniente para especificar a partição da lista L efetuada pelo Algoritmo Q^R. Denote por $\mathcal{X}'$ o conjunto de caixas em $\mathcal{R}^R(a,b)$ da seguinte maneira

$$\mathcal{X}'(a,b) = \left\{ c_i = (x_i, y_i, z_i) :\ y_i \leq b - \frac{b}{a}x_i \right\}\ .$$

Algoritmo $Q^R(L)$

Entrada: Lista de caixas L para o $PET^R(1,1)$.
Saída: Empacotamento $\wp$ de L em $B = (1, 1, \infty)$.

1. Rotacione as caixas de L e obtenha uma nova lista de caixas c tais que $x(c) \leq y(c)$. Chame de L a nova lista.

2. Divida L nas sublistas $L_1, \ldots, L_{10}$, conforme indicado a seguir (veja a Figura 4.4).

$$L_1 = L \bigcap \mathcal{C}\left[\tfrac{1}{2}, 1\ ;\ \tfrac{1}{2}, 1\right](1,1), \qquad L_2 = \left(L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{1}{2}\ ;\ \tfrac{1}{2}, 1\right](1,1)\right) \setminus \mathcal{X}'(1,1),$$
$$L_3 = L \bigcap \mathcal{C}\left[\tfrac{1}{4}, \tfrac{1}{3}\ ;\ \tfrac{2}{3}, 1\right](1,1), \qquad L_4 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ \tfrac{16}{27}, 1\right](1,1),$$
$$L_5 = L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{1}{2}\ ;\ \tfrac{1}{2}, \tfrac{2}{3}\right](1,1) \bigcap \mathcal{X}'(1,1), \qquad L_6 = L \bigcap \mathcal{C}\left[\tfrac{1}{4}, \tfrac{1}{3}\ ;\ \tfrac{1}{2}, \tfrac{2}{3}\right](1,1),$$
$$L_7 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ \tfrac{1}{2}, \tfrac{16}{27}\right](1,1), \qquad L_8 = L \bigcap \mathcal{C}\left[\tfrac{1}{3}, \tfrac{1}{2}\ ;\ \tfrac{1}{3}, \tfrac{1}{2}\right](1,1),$$
$$L_9 = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ \tfrac{1}{3}, \tfrac{1}{2}\right](1,1), \qquad L_{10} = L \bigcap \mathcal{C}\left[0, \tfrac{1}{3}\ ;\ \tfrac{1}{4}, \tfrac{1}{3}\right](1,1),$$
$$L_{11} = L \bigcap \mathcal{C}\left[0, \tfrac{1}{4}\ ;\ 0, \tfrac{1}{4}\right](1,1), .$$

3. $\wp_1 \leftarrow UC(L_1)$.

4. $\wp_i \leftarrow NFDH^x(L_i),\ i = 2, 3, 4, 8, 9, 10$.

5. Obtenha um empacotamento $\wp_5$ de L_5 da seguinte maneira .

5.1. Ordene L_5 em ordem não-crescente de altura .

5.2. Particione L_5 em sublistas $L_5^1, L_5^2, \ldots, L_5^{n_5}$ tais que

$$\begin{array}{lcl} L_5 & = & L_5^1 \| L_5^2 \| \ldots \| L_5^{n_5}, \\ |L_5^i| & = & 3, \quad i = 1, \ldots, n_5 - 1, \\ |L_5^{n_5}| & \leq & 3. \end{array}$$

5.3. Construa um empacotamento $\wp_5^i$ da lista L_5^i, $i = 1, \ldots, n_5$ como segue .

5.3.1. Se L_5^i tiver uma ou duas caixas, empacote essas caixas em um só nível. Seja $\wp_5^i$ este empacotamento.

5.3.2. Caso contrário,

5.3.2.1 Escolha $c \in L_5^i$, tal que $x(c)$ é mínimo .

5.3.2.2 Empacote as duas caixas de $L_5^i \setminus \{c\}$ nas posições $(0, 0)$ e $(\frac{1}{2}, 0)$. Empacote $\rho(c)$ na posição $(0, 1 - x(c))$.

5.3.2.3 Seja $\wp_5^i$ o empacotamento obtido .

5.4. $\wp \leftarrow \wp_5^1 \| \ldots \| \wp_5^{n_5}$.

6. Construa um empacotamento $\wp_6$ da seguinte maneira .

6.1. Ordene L_6 em ordem não-crescente de altura .

6.2. Particione L_6 em sublistas $L_6^1, L_6^2, \ldots, L_6^{n_6}$ tal que

$$\begin{array}{lcl} L_6 & = & L_6^1 \| L_6^2 \| \ldots \| L_6^{n_6} \ , \\ |L_6^i| & = & 4, \quad i = 1, \ldots, n_6 - 1 \ , \\ |L_6^{n_6}| & \leq & 4 \ . \end{array}$$

6.3. Construa um empacotamento $\wp_6^i$ da lista L_6^i, $i = 1, \ldots, n_6$ como segue .

6.3.1. Empacote três caixas de L_6^i nas posições $(0,0), (\frac{1}{3}, 0)$ e $(\frac{2}{3}, 0)$. Seja c a caixa de L_6^i ainda não empacotada. Empacote $\rho(c)$ na posição $(0, \frac{2}{3})$.

6.3.2. Seja $\wp_6^i$ o empacotamento gerado no passo anterior.

6.4. $\wp_6 \leftarrow \wp_6^1 \| \ldots \| \wp_6^{n_6}$.

7. Construa um empacotamento $\wp_7$ da seguinte maneira.

7.1. Ordene L_7 em ordem não-crescente de altura.

7.2. Construa os empacotamentos $\wp_7^1, \wp_7^2, \ldots$, nesta ordem, como segue.

7.3 Faça $i \leftarrow 1$.

7.2.1. Construa um empacotamento $\wp_7^i$ como segue.

7.2.1.1. Usando o Algoritmo $NFDH^x$, vá empacotando caixas de L_7 (ainda não empacotadas), enquanto for possível colocar caixas em um único nível.

7.2.1.2. Continue empacotando as caixas restantes com o Algoritmo $NFDH^y$, começando na posição $(1, \frac{16}{27})$, rotacionando cada caixa antes de empacotar, até que não seja mais possível empacotar novas caixas no mesmo nível ou não haja mais caixas a serem empacotadas.

7.2.1.3. Seja $\wp_7^i$ o empacotamento gerado nos passos 7.2.1.1 e 7.2.1.2 .

7.2.1.4. Faça $i \leftarrow i + 1$ e repita o passo 7.2.1 para o restante das caixas.

7.3. $\wp_7 \leftarrow \wp_7^1 \| \wp_7^2 \| \ldots$.

8. $\wp_{11} \leftarrow G_4(L_{11})$.

9. $\wp \leftarrow \wp_1 \| \ldots \| \wp_{11}$.

10. Retorne $\wp$.

fim algoritmo.

Teorema 4.5.1 *Para qualquer instância L do $PET^R(1,1)$, onde nenhuma caixa tem altura maior do que Z temos que*

$$Q^R(L) \leq 2,6875 \cdot OPT^R(L) + 10Z.$$

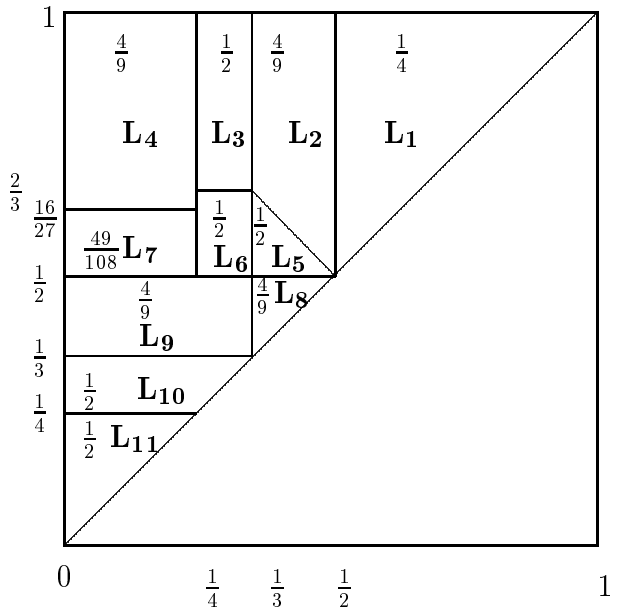


Figura 4.4: Partição da lista L efetuada pelo Algoritmo Q^R.

Dem. Pelos mesmos processos vistos antes, pode-se provar as seguintes desigualdades:

$$H(\wp_1) \leq 4V(L_1)\ , \tag{4.15}$$

$$H(\wp_i) \leq \frac{9}{4}V(L_i) + Z, \quad i = 2,3,4,8,9,10,11\ . \tag{4.16}$$

Analisaremos os empacotamentos $\wp_5$, $\wp_6$ e $\wp_7$ separadamente.

No empacotamento $\wp_5$, temos que a área garantida por nível, exceto talvez o último, é de $3 \cdot \frac{1}{6} = \frac{1}{2}$. Portanto, pelo Lema 3.2.2 concluimos que

$$\begin{aligned} H(\wp_5) &\leq 2V(L_5) + Z \\ &\leq \frac{9}{4}V(L_5) + Z. \end{aligned} \tag{4.17}$$

Note que se c é uma caixa de L_5^i escolhida no Passo 5.3.2, então todas as outras caixas de L_5^i têm largura menor que $1 - x(c)$.

No empacotamento $\wp_6$, é claro que é possível colocar quatro caixas por nível. Logo, a área garantida por nível, exceto talvez o último, é de $4 \cdot \frac{1}{4} \cdot \frac{1}{2} = \frac{1}{2}$. Novamente pelo Lema 3.2.2, concluimos que

$$\begin{aligned} H(\wp_6) &\leq 2V(L_6) + Z \\ &\leq \frac{9}{4}V(L_6) + Z\ . \end{aligned} \tag{4.18}$$

Finalmente, o empacotamento $\wp_7$ é a concatenação de um ou mais empacotamentos, cada qual construído em duas fases (cf. Passos 7.2.1.1 e 7.2.1.2).

Primeiramente é aplicado o Algoritmo $NFDH^x$, que empacota as caixas até que não se consiga colocar mais caixas em um mesmo nível. Assim, nesta primeira fase é garantida neste nível uma área de pelo menos $\frac{3}{8}$.

Em seguida, é aplicado o Algoritmo $NFDH^y$ na região $[0,1] \times [\frac{16}{27}, 1]$. É fácil ver que a área garantida é de pelo menos $(\frac{11}{27} - \frac{1}{4}) \cdot \frac{1}{2} = \frac{17}{216}$. Assim, a área garantida em cada nível de $\wp_7$, exceto talvez o último, é de pelo menos $\frac{17}{216} + \frac{3}{8} = \frac{49}{108}$. Portanto, pelo Lema 3.2.2, temos

$$\begin{aligned} H(\wp_7) &\leq \frac{108}{49}V(L_7) + Z \\ &\leq \frac{9}{4}V(L_7) + Z. \end{aligned} \tag{4.19}$$

Procedendo da mesma forma como nas demonstrações anteriores, obtemos que

$$Q(L) \leq \alpha \cdot OPT^R(L) + 10Z,$$

onde $\alpha = \frac{h+H}{max\{h, \frac{h}{4} + \frac{1}{3}H\}} \leq 2,6875$.

□

Apêndice A

Complexidade do Problema do Empacotamento Tridimensional

Neste apêndice veremos alguns resultados relativos à complexidade computacional do Problema do Empacotamento Tridimensional.

Complexidade no caso bidimensional

Mencionaremos nesta seção alguns resultados relativos ao caso bidimensional que serão usados para provar a complexidade computacional do PET e de seus casos especiais.

É fácil ver que o problema do empacotamento de retângulos em um retângulo é $\mathcal{NP}$-completo, já que este é uma generalização do caso unidimensional. O resultado para o caso unidimensional está provado em [10].

Lema A.0.2 *O problema do empacotamento de retângulos em em retângulo é $\mathcal{NP}$-completo.*

Mais ainda, Leung et al [20] provaram que o problema continua $\mathcal{NP}$-completo mesmo no caso especial em que todos os retângulos são quadrados.

Lema A.0.3 *O problema do empacotamento de quadrados em um quadrado é $\mathcal{NP}$-completo.*

Em 1990, Li e Cheng [23] provaram que mesmo quando os retângulos da lista são *pequenos* e quadrados, o problema continua $\mathcal{NP}$-completo.

Lema A.0.4 *Para qualquer constante $m \geq 1$, o problema do empacotamento de uma lista de quadrados $((x_1, x_1), \ldots, (x_n, x_n))$ em um retângulo (a, b) tal que $x_i \leq \frac{a}{m}$, $x_i \leq \frac{b}{m}$ é $\mathcal{NP}$-completo.*

Complexidade do PET

Como já visto no fim do Capítulo 2, o PET é $\mathcal{NP}$-difícil, já que este é uma generalização do caso bidimensional. Usando os resultados da seção anterior, podemos provar a complexidade do PET para os casos particulares vistos nesta dissertação. Os seguintes lemas podem ser provados usando os lemas da seção anterior.

Lema A.0.5 *O Problema do Empacotamento Tridimensional é $\mathcal{NP}$-difícil.*

Lema A.0.6 *O PET é $\mathcal{NP}$-difícil mesmo no caso especial em que as instâncias (L, a, b) do PET satisfazem as seguintes condições:*
(C1) $a = b$;
(C2) $L \in Q(a, b)$.

Lema A.0.7 *Qualquer que seja a constante $m \geq 1$, o PET é $\mathcal{NP}$-difícil mesmo no caso especial em que as instâncias $L = (c_1, \ldots, c_n)$ do $PET(a, b)$ satisfazem as seguintes condições:*
(C1) $x_i \leq \frac{a}{m}$ para $i = 1, \ldots, n$;
(C2) $y_i \leq \frac{b}{m}$ para $i = 1, \ldots, n$;
(C3) $x_i = y_i$ para $i = 1, \ldots, n$.

Mais ainda, Li e Cheng [23] mostraram que 2 é o limite inferior de desempenho absoluto de um algoritmo polinomial para os correspondentes problemas, a menos que $\mathcal{P} = \mathcal{NP}$.

Teorema A.0.8 *Nenhum algoritmo polinomial para o PET tem limite de desempenho absoluto $\alpha < 2$, a menos que $\mathcal{P} = \mathcal{NP}$.*

Dem. Vamos mostrar que se existir um algoritmo polinomial para o PET com tal limite de desempenho, então poderemos resolver em tempo polinomial o problema de decidir se uma dada lista de retângulos pode ser empacotada em um retângulo.

Suponha que exista um algoritmo polinomial $\mathcal{A}$ para o PET, com limite de desempenho absoluto $\alpha < 2$, i.e.,

$$\mathcal{A}(L) \leq \alpha \cdot OPT(L), \quad \text{para toda instância } L \text{ do } PET.$$

Seja $L' = ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$ uma instância qualquer do problema do empacotamento de retângulos em um retângulo (a, b). A partir de L' construímos uma instância L do $PET(a, b)$ tomando $L = ((x_1, y_1, 1), (x_2, y_2, 1), \ldots, (x_n, y_n, 1))$. Provaremos que L' é empacotável em (a, b) se e só se $\mathcal{A}(L) < 2$. Se L' é empacotável em (a, b), então temos que $OPT(L) = 1$. Como $\mathcal{A}(L) \leq \alpha \cdot OPT(L)$, segue que $\mathcal{A}(L) < 2$. Por outro lado, suponha que L' não é empacotável em (a, b). Então temos que, em qualquer empacotamento de L em (a, b, ∞), pelo menos uma caixa de L deve ficar acima de outra e portanto $OPT(L) \geq 2$. Como $\mathcal{A}(L) \geq OPT(L)$, temos que $\mathcal{A}(L) \geq 2$.

Como a redução feita é polinomial e o algoritmo $\mathcal{A}$ é polinomial, pelo Lema A.0.2, podemos concluir que $\mathcal{P} = \mathcal{NP}$.

□

Usando construções análogas às apresentadas na demonstração do Teorema A.0.8, podem-se demonstrar os seguintes teoremas.

Teorema A.0.9 *Nenhum algoritmo polinomial para o PET tem limite de desempenho absoluto $\alpha < 2$, a menos que $\mathcal{P} = \mathcal{NP}$, mesmo no caso especial em que as instâncias (L, a, b) do PET satisfazem as seguintes condições:*
(C1) $a = b$;
(C2) $L \in Q(a, b)$.

Dem. A demonstração deste teorema pode ser feita analogamente à demonstração do Teorema A.0.8, bastando para isso usar o Lema A.0.3.

□

Teorema A.0.10 *Qualquer que seja a constante $m \geq 1$, nenhum algoritmo polinomial para o $PET(a, b)$ tem limite de desempenho absoluto $\alpha < 2$, a menos que $\mathcal{P} = \mathcal{NP}$, mesmo no caso especial em que as instâncias $L = (c_1, \ldots, c_n)$ satisfazem as seguintes condições:*
(C1) $x_i \leq \frac{a}{m}$ para $i = 1, \ldots, n$;
(C2) $y_i \leq \frac{b}{m}$ para $i = 1, \ldots, n$;
(C3) $x_i = y_i$ para $i = 1, \ldots, n$.

Dem. Pode ser feita analogamente à demonstração do Teorema A.0.8, desta vez usando o Lema A.0.4.

□

Apêndice B

Problema do Empacotamento Tridimensional em Caixas

Neste apêndice apresentaremos um problema muito parecido com o PET, chamado **Problema do Empacotamento Tridimensional em Caixas**. Este problema pode ser assim definido:

Dados uma lista de caixas $L = (c_1, \ldots, c_n)$ e uma coleção de caixas $B = (a, b, c)$, empacotar L em caixas B de forma a minimizar o número de caixas B usadas no empacotamento

A intenção aqui é apenas a de fazer um breve comentário a respeito deste problema, devido à sua forte relação com o problema estudado nesta dissertação.

Veremos a seguir que as principais abordagens para este problema se dão em duas direções: abordagem combinatorial e abordagem experimental.

Abordagem Combinatorial

O estudo dos algoritmos que utilizam a abordagem combinatorial, onde as análises de desempenho dos algoritmos são semelhantes às estudadas nesta dissertação, ainda é recente e os limites de desempenho conhecidos são muito altos.

Em 1989, Coppersmith e Raghavan [7], desenvolveram um algoritmo *on-line* com limite de desempenho assintótico igual a 6,25. Este algoritmo usa o mesmo esquema de arredondamento definido para o algoritmo $\mathcal{A}LS_r^*$ visto no Capítulo 3.

Em 1992 Li e Cheng [24] usaram o mesmo esquema de arredondamento do algoritmo $FFLS_{r,s}$ (visto também no Capítulo 3) e obtiveram um algoritmo com limite de desempenho assintótico que pode se tornar tão próximo de 4,913 quanto se queira. Assim como

ocorreu no caso do algoritmo $FFLS_{r,s}$, o algoritmo desenvolvido apresenta alto valor para a constante aditiva do desempenho assintótico.

Em [21], Li e Cheng generalizaram o algoritmo H_M (visto no Capítulo 2) usando os mesmos esquemas de arredondamentos do algoritmo $FFLS_{r,s}$ conseguindo um algoritmo *on-line* com limite de desempenho assintótico que pode se tornar tão próximo de 4,84 quanto se queira. Este algoritmo também apresenta alto valor para a constante aditiva do desempenho assintótico.

Abordagem Experimental

Os algoritmos que fazem uso desta abordagem, usam em geral *métodos de programação linear* e são comumente chamados de **corte de estoque** (*cutting stock*).

Vejamos como o Problema do Empacotamento Tridimensional em Caixas pode ser formulado como um problema de programação linear inteira.

Dados uma lista de caixas distintas $L = (c_1, \ldots, c_n)$, inteiros $b_1, \ldots, b_n$, onde cada b_i representa o número de caixas c_i a serem empacotadas, e uma caixa $B = (a, b, c)$, considere padrões $\mathcal{P}_1, \ldots, \mathcal{P}_m$, onde cada padrão $\mathcal{P}_j$ representa uma forma de dispor caixas de L em B.

Dado um padrão $\mathcal{P}_j$, sejam $a_{1j}, a_{2j}, \ldots, a_{nj}$ inteiros, onde cada a_{ij} representa o número de caixas c_i no padrão $\mathcal{P}_j$. Seja x_j a variável que indica quantas vezes o padrão $\mathcal{P}_j$ é usado no empacotamento. Assim, o problema do Empacotamento Tridimensional em Caixas pode ser formulado como segue.

$$\text{minimizar } \sum_{j=1}^{m} x_j$$

$$\sum_{j=1}^{m} a_{ij} x_j = b_i \quad \text{para} \quad i = 1, \ldots, n \ .$$

$$x_j \geq 0, \quad x_j \ \text{ inteiro } \quad j = 1, \ldots, m \ .$$

Entretanto, há duas dificuldades:

- O problema acima é um problema de programação linear inteira, que é sabidamente $\mathcal{NP}$-difícil (veja Garey e Johnson [10]).

  Um método que pode gerar resultados satisfatórios, é achar uma solução ótima $x = (x_1, \ldots, x_m)$, não necessariamente inteira, e arredondar os valores x_j (para cima ou para baixo) de forma a suprir a necessidade de caixas a serem empacotadas.

- Outra dificuldade é o fato de que mesmo que o número de caixas distintas seja pequeno, a quantidade de padrões diferentes pode se tornar muito grande, de tal

forma que a quantidade de tempo e espaço necessários para tabular (sem mencionar a resolução do problema de programação linear) pode ser simplesmente inviável.

Uma forma engenhosa de contornar esta dificuldade foi sugerida por Gilmore e Gomory [11]. A técnica consiste em trabalhar com um número reduzido de padrões e gerar novos padrões somente quando estes forem necessários. Este método é chamado de **geração de colunas** [4].

O número de padrões possíveis também pode ser reduzido, eliminando-se padrões equivalentes ou restringindo-se os padrões para padrões do tipo **guilhotina** [12] (um padrão P é do tipo guilhotina se as caixas de P podem ser agrupadas de forma que sejam necessários apenas cortes paralelos à alguma face da caixa B).

Uma versão diferente foi tratada por Morábito e Arenales [27], na qual considera-se uma única caixa B objetivando maximizar o volume total das caixas empacotadas. Estes autores propõem uma abordagem que usa 'grafos-E/OU'. Segundo os autores, os resultados obtidos com esta abordagem produzem melhores soluções do que outros métodos conhecidos, demandando porém, maiores esforços computacionais.

Apêndice C

Aplicações

Empacotamento Unidimensional

- *Alocação de comerciais em TV.*

  Tem-se n comerciais para televisão $p_1, p_2, \ldots, p_n$, tendo cada comercial p_i uma duração de s_i segundos. Esses comerciais devem ser apresentados em intervalos de um programa de televisão, onde cada intervalo possui C segundos. Deseja-se alocar esses comerciais de modo a diminuir o número de intervalos necessários para apresentar esses comerciais.

- *Alocação de programas em discos e fitas magnéticas.*

  Programas de computador de tamanhos $p_1, p_2, \ldots, p_n$, cada programa p_i com tamanho s_i bytes, devem ser colocados em trilhas (ou setores) de discos magnéticos, de forma a usar o menor número de trilhas para gravar os n programas.

- *Carregamento de veículos.*

  Carregar n cargas com peso s_i, $(i = 1, \ldots, n)$, em veículos com limite de peso C, de maneira a não violar a restrição de lotação de cada veículo, com o objetivo de minimizar o número de veículos necessários.

- *Escalonamento de tarefas.*

  Minimizar o número de máquinas necessárias para completar n tarefas $p_1, p_2, \ldots, p_n$, em um dado limite de tempo C. Sabe-se que cada tarefa p_i requer um tempo s_i para ser executada.

- *Problema da programação de veículos.*

  Cargas (por ex., jornais) devem ser transportadas por veículos, a partir de um depósito até os pontos de entrega, em no máximo C horas. Cada veículo pode fazer

várias viagens. Programar n viagens com tempo de duração s_i horas ($i = 1, \ldots, n$), de maneira a não atrasar a entrega das cargas e com o objetivo de minimizar o número de veículos necessários.

- *Corte de bobinas (papel, aço, ...).*

  Bobinas (por ex., papel), de comprimento C são produzidas por uma fábrica e devem ser cortadas em diversos rolos de comprimentos $s_1, \ldots, s_n$, de forma a minimizar o número de bobinas necessárias.

- *Corte de vigas (em madeireiras, construção civil, ...).*

  Barras (por ex., ferro), de comprimento C, devem ser cortados em diversas barras menores de comprimentos $s_1, \ldots, s_n$. O objetivo é cortar o menor número possível de barras de comprimento C.

- *Pré-paginação.*

  Frações de páginas de memória (de computador) de tamanhos $s_1, \ldots, s_n$ devem ser alocadas em páginas de tamanho C bytes (frações de página são requeridas por segmentos de programas e estes devem aparecer em um menor número possível de páginas, por ex., *loops* internos, *arrays*). Encontrar um alocação que minimize o número de páginas de tamanho C.

Empacotamento Bidimensional em Placas

- *Corte de placas (vidro, chapas, madeira, ...).*

  Placas R de tamanho (a, b) devem ser cortadas em placas menores de vários tamanhos $(r_1, \ldots, r_n)$. O objetivo é minimizar o número de placas R necessárias para cortar as placas menores.

Empacotamento Bidimensional em Faixa

- *Corte de retalhos em fábrica de tecidos, ou em confecção de roupas.*

  Retalhos de tecidos retangulares $r_1, \ldots, r_n$ devem ser cortados em um rolo de tecido de largura a, objetivando-se minimizar o comprimento do rolo de tecido a ser cortado.

- *Corte de películas de filme (foto).*

  Uma película de filme de largura a deve ser cortada em n fotos de tamanhos retangulares, objetivando-se minimizar o comprimento da película de filme usada.

- *Escalonamento de tarefas em sistemas paralelos.*

  Um conjunto de n programas $r_1, \ldots, r_n$ devem ser executados dispondo se de uma quantidade de recurso a. Cada programa $r_i = (w_i, h_i)$ necessita de w_i unidades do recurso e leva h_i unidades de tempo para ser executado. O objetivo é minimizar o tempo necessário para executar os n programas.

Empacotamento Tridimensional (PET)

- *Empacotamento de caixas em galpões.*

  Caixas $c_1, \ldots, c_n$ devem ser armazenadas em um galpão com fundo $a \times b$, de forma a minimizar a altura da disposição final das caixas.

Empacotamento Tridimensional com Rotação (PET^R)

- *Escalonamento de tarefas em sistemas particionáveis de malha conexa (job scheduling in partitionable mesh connected systems).*

  Um computador paralelo com uma configuração que forma uma malha de processadores de forma retangular (em geral quadrada) deve executar n processos $c_1, \ldots, c_n$, onde cada processo $c_i = (x_i, y_i, z_i)$ usa uma submalha de tamanho (x_i, y_i) gastando z_i unidades de tempo. O objetivo aqui é alocar os processos de modo a minimizar o tempo total para que o computador execute os n processos.

Empacotamento Tridimensional em Caixas

- *Empacotamento de contêineres.*

  Caixas $c_1, \ldots, c_n$ devem ser empacotadas em *containers* de tamanho (a, b, c). O objetivo é fazer o empacotamento de modo a minimizar o número de *containers* usados.

- *Carregamento de cargas em furgões.*

  Carregar cargas $c_1, \ldots, c_n$ em veículos de transporte com dimensões de (a, b, c). O objetivo aqui é minimizar o número de veículos necessários para transportar as n cargas.

Apêndice D

Tabela dos algoritmos e seus limites de desempenho assintótico

Nas tabelas apresentadas a seguir indicamos os algoritmos mencionados nesta dissertação e os valores α e β relativos aos limites de desempenho assintótico desses algoritmos. Para cada algoritmo $\mathcal{A}$, temos que α e β são tais que $\mathcal{A}(L) \leq \alpha \cdot OPT(L) + \beta Z$ para toda lista L onde nenhuma caixa tem altura maior que Z.

Os algoritmos que desenvolvemos estão precedidos do símbolo $\star$.

Algoritmos para o $PET(a, b)$

Caso Geral

Algoritmo	α	β	**Página**
G_1	$4,57$	2	42
$NFLS_r$	$\lim_{r\to 1} \alpha = 4$	$\frac{3}{r(1-r)}$	85
$NFLS_r^*$	$\lim_{r\to 1} \alpha = 3,5$	$\frac{14}{3}\frac{1}{r(1-r)}$	89
$FFLS_r$, $BFLS_r$	$\lim_{r\to 1} \alpha = 3,4$	$\frac{5}{r(1-r)}$	85
$H_M LS_r$	$\lim_{r\to 1, M\to\infty} \alpha = 3,382$	$\frac{1+2(M-1)}{r(1-r)}$	85
C	$3,25$	7	49
$\star\, R$	$3,04904$	14	69
$\star\, T$	3	$\frac{141}{8}$	76
$FFLS_r^*$, $BFLS_r^*$	$\lim_{r\to 1} \alpha = 2,975$	$\frac{22}{3}\frac{1}{r(1-r)}$	89
$H_M LS_r^*$	$\lim_{r\to 1, M\to\infty} \alpha = 2,95925$	$\frac{2+\frac{8}{3}(M-1)}{r(1-r)}$	89
$FFLS_{r,s}$	$\lim_{r\to 1, s\to 1} \alpha = 2,89$	$O\left(\frac{1}{r(1-r)}\frac{1}{s(1-s)}\right)$	94

Para caixas pequenas $(L \in \mathcal{R}_m(a,b))$

Algoritmo	α	β	Condição	Página
G_m	$\frac{m}{m-2}$	1	$m \geq 3$	45
C_m	$\frac{m+1}{m-1}$	3	$m \geq 2$	66
$\star\ C_m^*$	$\left(\frac{m+1}{m}\right)^2$	6	$m \geq 2$	67

Para caixas com fundo quadrado $(L \in \mathcal{Q}(a,b))$

Algoritmo	α	β	Página
GQ_1	4	1	47
$\star\ CQ$	2,796875	$\frac{77}{8}$	61

Para caixas com fundo pequeno e quadrado $(L \in \mathcal{Q}_m(a,b))$

Algoritmo	α	β	Condição	Página
GQ_m	$\left(\frac{m}{m-1}\right)^2$	1	$m \geq 2$	47
CQ_m	$\left(\frac{m+1}{m}\right)^2$	1	$m \geq 3$	59

Para $a = b$ **e caixas com fundo quadrado** $(L \in \mathcal{Q}(a,a))$

Algoritmo	α	β	Página
CQQ	2,6875	2	57

Algoritmos para o $PET^R(a,b)$

Algoritmo	α	β	Condição	Página
$\star\ R^R$	3,04904	15		102
$\star\ P^R$	3	9	$max\{x_i, y_i\} \leq min\{a,b\}$	105
$\star\ Q^R$	2,6875	10	$a = b$	109

Considerações finais

Chegamos ao final desta dissertação, mas a sensação que temos é de que há muito ainda o que explorar a respeito dos tópicos aqui abordados. O Problema do Empacotamento Tridimensional – tanto na versão orientada como na versão com rotação do fundo – só passou a ser investigado recentemente, razão pela qual não se pode considerar exauridas as abordagens a serem tentadas.

Vimos no Capítulo 3 que a idéia básica das estratégias desenvolvidas por Li e Cheng [22] reside na partição da lista dada em sublistas, de modo que algoritmos apropriados para cada uma destas sublistas possam ser usados. Vimos também que é possível obter algoritmos com limites de desempenho melhores quando combinamos as sublistas mais 'críticas', de modo a obter uma melhor garantia de área ocupada. Parece-nos que esta estratégia de combinar sublistas críticas pode ser melhor explorada, fazendo uma análise mais cuidadosa das possíveis combinações de sublistas.

Notamos também que muitos algoritmos para o caso tridimensional foram construídos a partir de generalizações dos casos unidimensional e bidimensional. Assim, uma pergunta que surge é se outros algoritmos dos casos unidimensional e bidimensional poderiam ser generalizados para o caso tridimensional, garantindo limites de desempenho assintótico melhores.

Para o cálculo dos limites de desempenho assintótico dos algoritmos apresentados, foram utilizadas relações entre a altura do empacotamento de uma dada sublista e a altura do empacotamento ótimo da lista. Para fazer isso, várias abordagens foram usadas, algumas usando comparações com volume e outras usando resultados de algoritmos unidimensionais. Apresentamos também uma nova abordagem usando um algoritmo de empacotamento bidimensional. Assim, uma outra pergunta que surge é se existem novas abordagens para obter alguma informação sobre a altura de um empacotamento ótimo de uma dada lista.

Os estudos que realizamos até o momento culminaram nesta dissertação, mas continuamos buscando respostas para estas e outras perguntas. Esperamos que o conhecimento adquirido ao elaborar esta dissertação possa ser usado na busca de outros resultados melhores.

Lista de símbolos

Bibliografia

[1] B. S. Baker. A new proof for the first-fit decreasing bin-packing algorithm. *J. of Algorithms*, 6:49–70, 1985.

[2] B. S. Baker, D. J. Brown, and H. P. Katseff. A $\frac{5}{4}$ algorithm for two-dimensional packing. *J. of Algorithms*, 2:348–368, 1981.

[3] B. S. Baker, E. G. Coffman Jr., and R. L. Rivest. Orthogonal packings in two-dimensions. *SIAM J. Comput.*, 9:846–855, 1980.

[4] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1980.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing - an updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithms design for computer system design*, pages 49–106. Spring-Verlag, New York, 1984.

[6] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level oriented two-dimensional packing algorithms. *SIAM J. Comput.*, 9:808–826, 1980.

[7] D. Coppersmith and P. Raghavan. Multidimensional on-line bin packing: Algorithms and worst-case analysis. *Oper. Res. Lett.*, 8(1):17–20, 1989.

[8] P. Erdös and R. L. Graham. On packing squares with equal squares. *J. Combinatorial Theory Ser. A*, 19:119–123, 1975.

[9] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao. Resource constrained scheduling as generalized bin packing. *J. Combinatorial Theory Ser. A*, 21:257–298, 1976.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Co., San Fransisco, 1979.

[11] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. *Ops. Res.*, 9:849–859, 1961.

[12] P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *Ops. Res.*, 13:94–120, 1965.

[13] D. S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1973.

[14] D. S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8:272–314, 1974.

[15] D. S. Johnson, A. Demars, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds form simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:299–325, 1974.

[16] D. S. Johnson and M. R. Garey. A $\frac{71}{60}$ theorem for bin packing. *J. Complexity*, 1:65–106, 1985.

[17] N. Karmakar and R. M. Karp. An efficient approximation scheme for the one dimensional bin packing problem. In *Proceedings, 23rd Ann. Symp. on Foundations of Computer Science*, pages 312–320, Los Angeles, 1982. IEEE Computer Society.

[18] R. M. Karp. *Reducibility among combinatorial problems*, pages 85–103. R. E. Miller and J. M. Thatcher, New York, 1972.

[19] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. Association Comput. Mach.*, 32:562–572, 1985.

[20] J. Y-T. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. L. Chin. Packing squares into squares. *J. Parallel and Distributed Computing*, 10:271–275, 1990.

[21] K. Li and K-H. Cheng. A generalized harmonic algorithm for on-line multidimensional bin packing. TR UH-CS-90-2, University of Houston, January 1990.

[22] K. Li and K-H. Cheng. On three-dimensional packing. *SIAM J. Comput.*, 19:847–867, 1990.

[23] K. Li and K-H. Cheng. Static job scheduling in partitionable mesh connected systems. *J. Parallel and Distributed Computing*, 10:152–159, 1990.

[24] K. Li and K-H. Cheng. Generalized first-fit algorithms in two and three dimensions. *Int. J. Found. Comput Sci.*, 1(2):131–150, 1992.

[25] K. Li and K-H. Cheng. Heuristic algorithms for on-line packing in three dimensions. *J. of Algorithms*, 13:589–605, 1992.

[26] A. Meir and L. Moser. On packing of squares and cubes. *J. Combinatorial Theory Ser. A*, 5:116–127, 1968.

[27] R. Morábito and M. Arenales. Uma abordagem em Grafo-E/OU para o Problema do Carregamento de Contêineres. In *Anais do XXV Simpósio Brasileiro de Pesquisa Operacional*, Campinas-SP, 1993. Unicamp.

[28] P. Ramanam, D. J. Brown, C. C. Lee, and D. T. Lee. On-line bin packing in linear time. *J. of Algorithms*, 10(3):305–326, 1989.

[29] A. C. Yao. New algorithms for bin packing. *J. Association Comput. Mach.*, 27:207–227, 1980.