CONTRIBUIÇÕES À SOLUÇÃO DO PROBLEMA BIN–PACKING:
FORMULAÇÕES, RELAXAÇÕES E NOVOS ALGORITMOS APROXIMATIVOS.

Dario José Aloise

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Nelson Maculan Filho, D.Sc.
(Presidente)

Prof. Ruy Eduardo Campello, D.Sc.

Prof. Jayme Luiz Szwarcfiter, Ph.D.

Profa. Nair Maria M. de Abreu, D.Sc.

Prof. Antônio Clécio F. Thomaz, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

ABRIL DE 1992

ALOISE, DARIO JOSÉ

Contribuições à Solução do Problema Bin–Packing:

Formulações, Relaxações e Novos Algoritmos Aproximativos

[Rio de Janeiro] 1992

IX, 165 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 1992)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Bin–Packing 2. Algoritmos Aproximativos 3. NP–completo

I. COPPE/UFRJ II. Título (Série).

À minha esposa Valmira

e nossos filhos,

Daniel e Davi

AGRADECIMENTOS

Ao **Prof. Nelson Maculan Filho** pela sua paciente, amiga e sábia orientação.

Ao **Prof. Ruy Eduardo Campello** por me ter sugerido a exploração deste tema como tese e, pelo seu estímulo, sugestões e conversas amigas sobre aspectos deste trabalho.

Ao **Prof. e colega de Departamento Carlos Augusto C. de Lima** pela valiosa ajuda no desenvolvimento do software da Árvore B–Tree, que serviu de suporte para a implementação de vários algoritmos.

Ao **ILTC/RJ**, por me colocar à disposição suas instalações e serviços.

À **UFRN**, em particular ao **Departamento de Informática e Matemática Aplicada**, pela minha liberação.

À **CAPES** e a **FAPERJ/RJ** pela concessão de bolsa de estudo.

À todos aqueles que de uma forma ou de outra colaboraram para a realização deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requesitos necessários para a obtenção do grau de Doutor em Ciências (D. Sc.).

CONTRIBUIÇÕES À SOLUÇÃO DO PROBLEMA BIN–PACKING: FORMULAÇÕES, RELAXAÇÕES E NOVOS ALGORITMOS APROXIMATIVOS.

Dario José Aloise

Abril de 1992

Orientador: Nelson Maculan Filho

Programa: Engenharia de Sistemas e Computação

O problema Bin–Packing é um problema combinatório de grande importância teórica e prática. Surgido no início da década de 70, tem sido tratado por vários cientistas e pesquisadores e uma quantidade considerável de artigos em revistas importantes vem sendo publicada. Porém quase nada ainda foi escrito em língua portuguesa sobre o referido problema. Aqui, o problema Bin–Packing é analisado.

Inicialmente este problema é enquadrado no contexto mais amplo de Cutting & Packing e uma abordagem exata do problema é apresentada. Por se tratar de um problema NP–árduo, uma atenção especial aos algoritmos aproximativos e suas performances, não poderia ser omitida. Como resultado, uma classe de algoritmos aproximativos decrescentes *off–line* foi desenvolvida e suas performances avaliadas com relação às heurísticas famosas, como First Fit Decreasing e Next Fit Decreasing. Uma extensão natural para o caso bi–dimensional do algoritmo chave dessa classe, denominado Maiores e Menores Decrescente foi também desenvolvida. Tais algoritmos apresentaram–se eficientes com o crescente número de ítens. Por último, foi sugerida algumas idéias para aplicação dos algoritmos a uma situação particular de empacotamento de caixas na indústria textil.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

CONTRIBUTIONS TO THE SOLUTION OF THE BIN–PACKING PROBLEM: FORMULATIONS, RELAXATIONS AND NEW APROXIMATIVE ALGORITHMS.

Dario José Aloise

April 1992

Thesis Supervisor: Nelson Maculan Filho

Department: Systems Engineering and Computer Science

This dissertation deals with the Bin–Packing a combinatorial optimization problem of great practical and theoretical importance. This problem has been brought to light in the early 70's and despite the huge amount of published papers found in the literature almost nothing has been written in Portuguese.

The problem is characterized in the wide context of the Cutting & Packing and exact algorithmic approach is given. Since it is an NP–hard problem special attention is given to the design and analysis of heuristics. This results in the research of a class of decreasing aproximative algorithms *off–line* and their performance compared to famous heuristcs such as First Fit Decreasing and Next Fit Decreasing. A natural extention to bi–dimensional of this class' key algorithm, named Maiores e Menores Decrescentes. This algorithms are quite eficient when the number of items is increased. Finally, we introduce some ideas for the use of this algorhtms to particular situation of packing Textil Industry boxes.

ÍNDICE

Capítulo I
Introdução

Desde o início da década de 70 tem havido um grande interesse na análise de modelos combinatoriais de problemas BIN–PACKING [42,43,44] e uma impressionante quantidade de pesquisa sobre este assunto tem sido publicada na literatura desde então quase todas tratando de algoritmos heurísticos (Cf. survey de Coffman, Garey & Johnson [14], o qual inclui uma bibliografia de mais do que uma centena de títulos publicados somente até 1984). Originalmente o problema clássico, uni–dimensional, objetiva minimizar o número de "bins" (i.e., regiões bem definidas, como: caixas, vagões, trilhas de um disco rígido, etc) de igual capacidade C necessários para o acomodamento de uma dada lista L de objetos, de magnitudes (e.g., de *dimensão espacial*: tamanho, espessura, etc; de *dimensão não–espacial*: tempo, palavra de computador, etc) nunca superiores a C. Este problema, ou mais precisamente o problema de decisão "Dados C e L, e um número inteiro k, pode L ser empacotado dentro de k ou menos bins de capacidade C ?" é NP–árduo no sentido **forte** (veja Garey & Johnson [28], p. 124), e assim é pouco provável que exista até mesmo um algoritmo de otimização em tempo pseudo–polinomial para ele, a não ser que $\mathcal{P} = \mathcal{NP}$. Por isso vários algoritmos aproximativos simples e rápidos que constroem empacotamentos próximos do ótimo têm sido propostos e estudados. Uma variante consiste em fixar o número de bins e maximizar a quantidade de objetos empacotados. Muitas outras variantes e generalizações foram obtidas para este problema ampliando consideravelmente o escopo de aplicações. Algoritmos Aproximativos têm sido construídos e analisados também para as seguintes quatro modificações básicas: (1) Empacotamentos nos quais um limite é fixado a priori no número de ítens que podem ser empacotados num bin; (2) Empacotamentos nos quais uma ordem parcial é associada com o conjunto de ítens a ser empacotado e restringe a maneira pela qual os ítens podem ser empacotados; (3) Empacotamentos nos quais restrições são colocadas nos ítens que podem ser colocados

num mesmo bin, e (4) Empacotamentos nos quais os ítens podem entrar e sair num bin dinâmicamente.

O interesse por este problema aparece por sua aplicação em diversas situações práticas, como **alocação de dados em computação** (p. ex., alocação de arquivos de dados em trilhas de um disco rígido; *prepaging* – frações de uma página representando segmentos de programas dentro de páginas de memória; *table formatting* – empacotamento de "strings" de comprimento variável dentro de palavras de comprimento fixado, etc.); em **cutting–stock** (p. ex., cortes de vergalhões usados na construção civil; de pedaços de canos ou tubos; de tecidos; de vidros, etc) e **programação de eventos** (ordenação de tarefas independentes, para minimizar o número de máquinas necessárias para completar todas as tarefas satisfazendo um dado limite de tempo "deadline", em um sistema de processadores paralelos – **computação paralela** – vizando minimizar o número de processadores ou minimizar o "makespan", etc). Tantas outras aplicações comerciais (como carregamento de caminhões, designação de programas em estações de televisão, etc.) podem ser encontradas em BROWN [6].

Esta dissertação é organizada como segue: O Capítulo I, contém uma breve introdução sobre o desenvolvimento e interesse pelo problema. No Capítulo II, resumidamente fazemos uma revisão de conceitos sobre a terminologia associada ao estudo de modelos combinatoriais mostrando em seguida a posição deste problema dentro do contexto da classe geral de problemas de Cutting & Packing. A partir daí, uma abordagem exata do problema clássico de Bin–Packing é iniciada pelos primeiros modelos desenvolvidos e culminando com um **modelo matemático generalizado** de bin packing, baseado no trabalho de LEWIS [53]. Diversas técnicas exatas conhecidas na literatura são abordadas, tais como Programação Linear Generalizada, o método Branch–and–Bound e Relaxação Lagrangeana que emprega a teoria de funções não–diferenciáveis no problema dual Lagrangeano semelhantemente como já ocorreu com outros problemas de otimização combinatória. Isto se justifica aqui, porque muito pouco se tem escrito sobre a obtenção da solução exata. De nosso conhecimento, somente os trabalhos de EILON & CHRISTOFIDES [22], HUNG & BROWN [38] e MARTELLO & TOTH [58]

para o problema Clássico, e LEWIS & PARK [54] para o modelo generalizado atendendo algumas variantes, fornecem procedimentos exatos.

No Capítulo III, fazemos uma revisão da terminologia empregada no estudo de algoritmos heurísticos para problemas NP–completo, definindo o que significam termos como "algoritmos aproximativos", "garantia de performance", "comportamento no pior–caso", "esquema de aproximação", etc. Em seguida, fixamos inicialmente a atenção no problema clássico e nas heurísticas básicas, desenvolvidas por D. S. JOHNSON *et al.* [44], que serviram de suporte para a consolidação e validade do emprego de algorimos aproximativos polinomiais em problemas de otimização NP–árduos. A prova do limite de performance do pior–caso para o conhecido algoritmo First–Fit usa como suporte uma função de peso e possui um conteúdo matemático um tanto complexo, e assim uma melhoria de entendimento, pela consulta em diversos artigos que tratam deste assunto, é apresentada, pois conforme cita COFFMAN [11] *"é comum escutar reclamações que, embora as funções de peso sejam capazes de verificar a correção na prova de limites assintóticos, suas origens como também a idéia das provas, ficam envolvidas em mistério..."*. Menciona–se ao final do capítulo, os resultados mais recentes na análise de pior–caso para o problema uni–dimensional; as classes gerais de problemas de empacotamento e suas variantes visando mostrar o desenvolvimento do estudo deste problema para o caso unidimensional.

A grande maioria dos algoritmos aproximativos de bin–packing, com raras exceções (e.g., algoritmo Next Fit Decreasing, First Fit Increasing), trabalham considerando um preenchimento simultâneo dos bins, ou seja os bins somente são considerados completos quando não existir mais nenhum elemento na lista para ser alocado. Ás vezes, no entanto, principalmente em produção em série, torna–se preferível que o empacotamento seja *on–line* em relação aos bins, isto porque enquanto um novo bin está sendo preenchido, o anterior já está tomando outro destino (p. ex., sendo armazenado). No Capítulo IV, propomos uma classe de algoritmos aproximativos, do tipo *off–line* em relação a lista de entrada e *on–line* em relação aos bins, para o problema clássico, originada de um algoritmo mais simples o qual denominamos de

MMD (**M**aiores e **M**enores **D**ecrescente). Um algoritmo Iterativo MMD é construído e seu desempenho estudado, para atender a variante do problema que fixa os bins e maximiza o número de elementos empacotados. Ao final do mesmo comentamos a respeito de um software desenvolvido para cópias eficientes de disquetes a partir de um disco rígido.

O Capítulo V é uma pequena revisão do problema Multi–Dimensional, sendo grande parte relacionada com o caso Bi–Dimensional, especificamente aos algoritmos 2D *off–line*. Neste, um algoritmo *BI–MMD* é desenvolvido como extensão natural do mesmo algoritmo uni–dimensional, sendo idealizado para aplicações onde possam ou não ser permitidas uma rotação de 90° nos ítens retangulares. Os resultados da qualidade do algoritmo é feito empiricamente sobre dados gerados aleatoriamente. Por último, abordamos uma aplicação prática 3D para empilhamento de caixas em cima de lastros de caminhões (modelo aplicado à Indústria Textil).

O Capítulo VI faz a conclusão do trabalho, oferecendo em adição sugestões para futuras pesquisas. Em Anexo, a demonstração das propriedades da função de peso $w(x)$ apresentadas na prova do teorema 2.

Capítulo II
O Problema Bin Packing : Uma Abordagem Exata

2.1 Problema de Otimização

Um *problema de otimização combinatória* P é *um problema de maximização* ou *de minimização* caracterizado por uma terna $P = (E_P, S_P, m_P)$, onde:

E_P é um conjunto de instâncias de P;

S_P é um mapeamento sobrejetor que associa a uma instância $I \in E_P$ um conjunto finito $S_P(I)$ de todas as candidatas a solução (i.e., soluções viáveis) para I;

m_P é uma função que atribui a cada instância $I \in E_P$ e cada candidato a solução $\sigma \in S_P(I)$ um número racional positivo $m_P(I,\sigma)$ chamado *valor solução* para σ.

Para cada $I \in E_P$ o valor ótimo m_P^* é definido por

$$m_P^* = MELHOR \{ m_P(I,\sigma) : \sigma \in S_P(I) \}$$

onde *MELHOR* pode ser *MAX* ou *MIN* dependendo se o problema é de maximização ou de minimização. Dado que $S_P(I)$ é finito, deve existir ao menos uma solução $\sigma^* \in S_P(I)$ tal que $m_P(I,\sigma^*) = m_P^*$, e tal solução é chamada *solução ótima*. Denominaremos, no restante do trabalho, $OPT(I)$ a $m_P(I,\sigma^*)$.

Constitui um problema combinatorial, qualquer problema de programação inteira com a seguinte formulação:

$$(PPI) \qquad \text{Minimizar } z = \boldsymbol{c}.x$$

s.a.

$$\boldsymbol{A}x \leq \boldsymbol{b}$$

$$\boldsymbol{D}x = \boldsymbol{e}$$

$$x \in \{0,1\}^n$$

onde $\boldsymbol{A}$ é uma matriz mxn, $\boldsymbol{D}$ é uma matriz kxn e $\boldsymbol{c}$, $\boldsymbol{b}$ e $\boldsymbol{e}$ são vetores de dimensões compatíveis. Assim dada uma instância I de (PPI) a todo elemento de $S_p(I)$ corresponde uma solução inteira.

Para a maioria dos problemas combinatoriais formulados desta maneira não existe, ou não se conhece, algoritmo **limitado polinomialmente** que ache uma solução $\sigma \in S_P$ com valor objetivo $z(\sigma)$ tal que tenhamos para todas as instâncias do problema

$$\left| \frac{z^* - z(\sigma)}{z^*} \right| < \varepsilon \,, \quad \forall\, \varepsilon > 0$$

(SAHNI & GONZALES [65]). Tais problemas são denominados NP–árduos (Fig. II.1), i.e, o esforço computacional que o algoritmo faz para encontrar a solução exata não é menor do que uma exponencial no tempo.

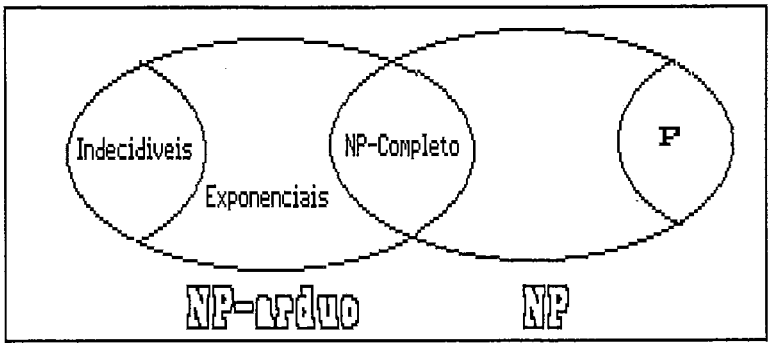


FIG. II.1: *Classes de Problemas.*

2.2 O Bin Packing Clássico

Formalmente o problema clássico uni–dimensional pode ser estabelecido, como um <u>problema de otimização</u>, da seguinte maneira: Dado um conjunto finito ou lista de ítens $L = \{a_1, a_2, \ldots, a_n\}$ juntamente com uma função s a qual associa a cada $a_i \in L$ um valor (p. ex., tamanho do ítem) $s(a_i)$ obdecendo $0 < s(a_i) \leq C$, e $B_1, B_2, \ldots$, uma seqüência infinita de bins ("mochilas") de capacidade comum igual a C. Determinar o menor inteiro k tal que exista uma partição $B = \{C_1, C_2, \ldots, C_k\}$ de L, i.e., $L = C_1 \cup C_2 \cup \ldots \cup C_k$, $C_i \cap C_j = \emptyset$ $\forall i \neq j$, satisfazendo $\Sigma_{a \in C_j} s(a) \leq C$ para cada j, $1 \leq j \leq k$. Cada conjunto C_j é pensado como sendo o conteúdo de um bin B_j, e como se vê a intenção é minimizar o número de bins necessários para acomodar os n ítens de L, sem sobreposição.

Como um <u>Problema de Decisão</u>: Dado L e um inteiro k. Pode L ser empacotado dentro de k ou menos bins ?

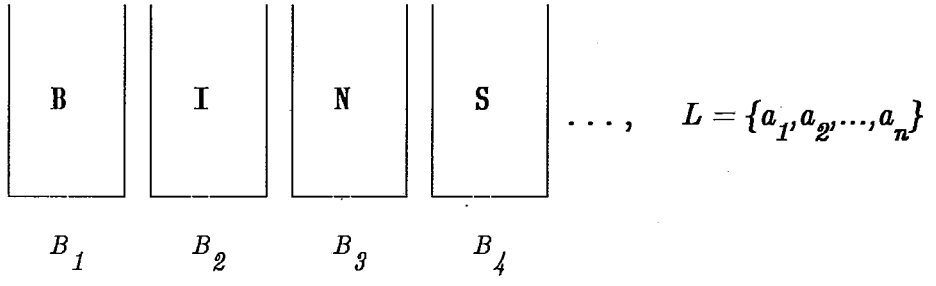


FIG. II.2: *O Problema Bin–Packing Clássico*

Em termos abstratos, e para fins de análise matemática, o problema pode ser estabelecido com o seguinte enunciado:

Dada uma lista L de n números reais em (0,1], colocar os elementos de L em um número mínimo L^* de "bins" de tal forma que nenhum bin contenha números cuja soma exceda 1.

Neste trabalho, como é usual, simplificaremos o formalismo usando sempre os mesmos símbolos para ítens e seus tamanhos.

O problema Bin–Packing modela uma variedade de situações do mundo real. Alguns exemplos comumente citados são:

Em ciência da computação (D. S. JOHNSON *et al.* [44]):

- **Table formatting**. Os bins são palavras de computador de tamanho fixado em k bits. Supondo que existam ítens de dados (p. ex., bit de string de comprimento *6*, caracter de string de comprimento *3* bytes, inteiro de meia–palavra) requerendo ka_1, ka_2,..., ka_n bits, respectivamente. Deseja–se colocar os dados no menor número possível de palavras.

- **Prepaging**. Os bins, agora, representam páginas e os números na lista L representam frações de uma página requerida por segmentos de programas os quais deveriam aparecer numa única página, p. ex., loops internos, arrays, etc.

- **Alocação de arquivos**. Arquivos inteiros de vários tamanhos são colocados, sem quebra, no mínimo possível de trilhas de um disco rígido.

Na Indústria (BROWN [6], CHVATAL [9]):

- **cutting stock uni–dimensional**, p. ex., cortes de canos, tubos, tábuas, papel, tecidos, etc.). Suponha, por exemplo, que um cliente necessite de canos de tamanhos diferentes $L = \{a_1, a_2, ..., a_n\}$ e o armazém local só dispõe de canos de tamanho padrão (i.e., bins). A encomenda seria comprar o menor número possível de canos padrões para cortá–los a fim de atender a demanda L (veja Fig. II.3).

- **Ordenação de tarefas independentes (scheduling)**. Os bins representam máquinas ou processadores *(computação paralela)* e C é o limite máximo de tempo pelo qual todas as tarefas devem ser completadas. $s(a_i)$ é o tempo de processamento requerido pela tarefa a_i. O problema é determinar o menor número de processadores necessários para executá–las dentro de um tempo limite estabelecido *(deadline)*.

- **Carregamento de veículos** (determinação do número) com um dado limite

de peso e **designação de comerciais** para intervalos em estação de televisão.

Todas estas situações podem ser igualmente vistas pelo exemplo da Fig. II.3:

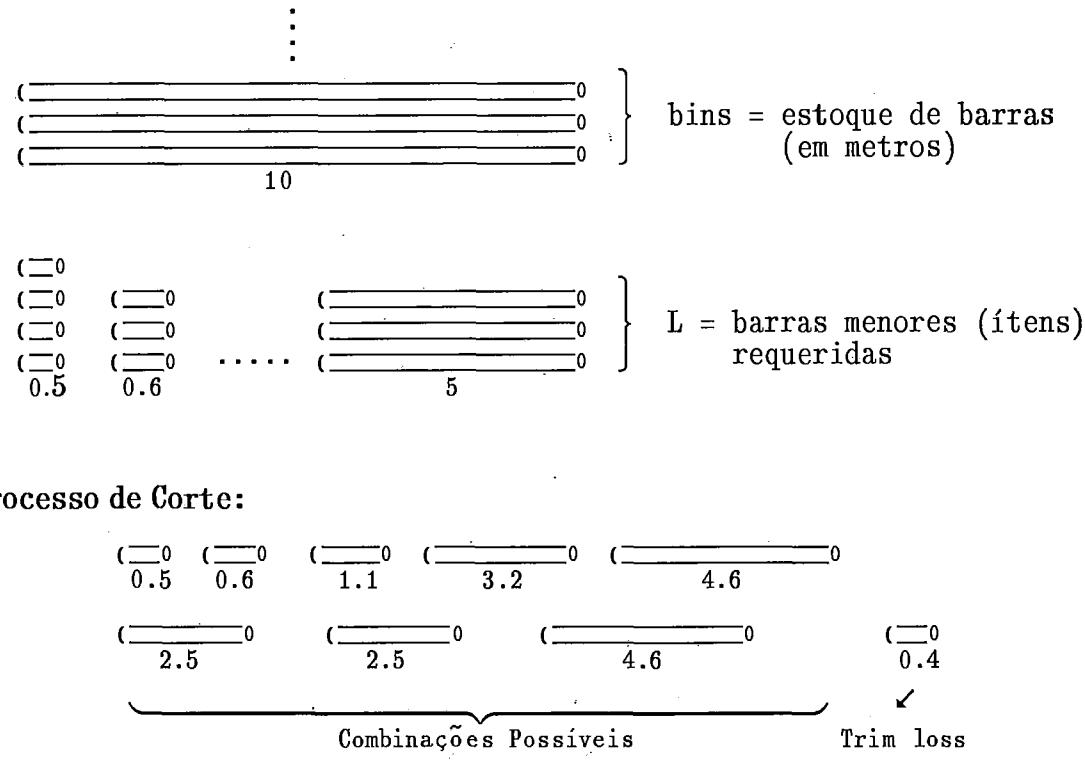


FIG. II.3: *Cutting–stock*

Infelizmente, o problema **Bin–Packing** (*BP*) é NP–árduo no **sentido forte** (transformação de 3–PARTITION, cf. GAREY & JOHNSON [28]) o que significa que nem mesmo um algoritmo de otimização em tempo pseudo–polinomial pode ser achado a menos que $\mathcal{P} = \mathcal{NP}$.

2.2.1 Enquadramento dentro do Contexto de Problemas de Cutting e Packing

O termo **bin packing** foi introduzido, por D. S. JOHNSON em [43], em substituição a **loading**, para tratar da designação de arquivos de dados de determinados

comprimentos às trilhas de um disco rígido. A partir daí tornou–se moda usar este termo para vários outros problemas relacionados a **packing**, principalmente quando se faz um estudo de heurísticas. Por exemplo, vários artigos sobre *BP* tratam de assuntos da área de scheduling, assembly line balancing, memory allocation e cutting–stock, e devido a forte relação entre eles, quando são vistos sob a ótica de empacotamento, são rotulados como bin packing, variantes ou generalizações do mesmo.

Mais recentemente DYCKHOFF [22], fez um diagrama relacional (Fig. II.4) de problemas de cutting e packing (C&P), que são bastante usados na literatura sob diferentes nomes, embora possuam a mesma estrutura lógica (Tabela 1). Este estudo procura integrar as várias espécies de problemas e unificar o uso de conceitos que são empregados indistintamente. Enquadra–se neste contexto o problema de Bin Packing também referido como dual bin packing, strip packing, vetor packing e knapsack (packing).

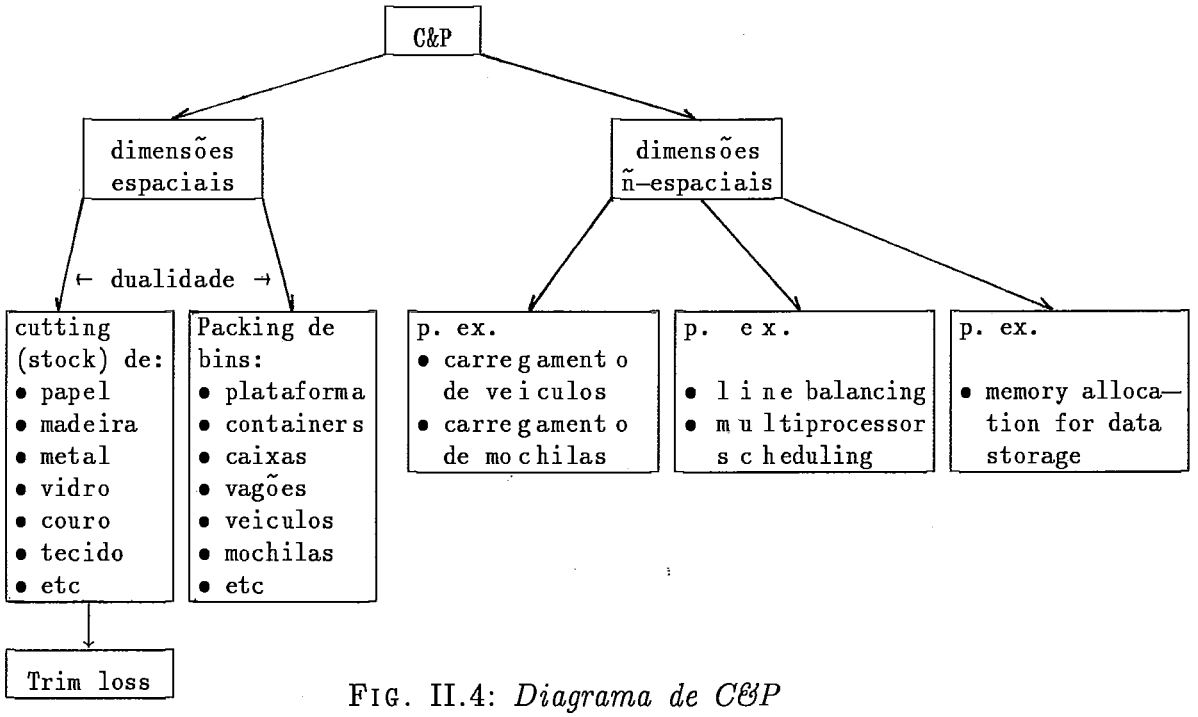


FIG. II.4: *Diagrama de C&P*

Para mostrar resumidamente, a forte relação existente entre alguns problemas de C&P, sejam as seguintes 4 características principais com seus respectivos tipos mais comumente citados denotados com os símbolos indicados:

1. Dimensionalidade
(1) Uni–dimensional.
(2) Bi–dimensional.
(3) Tri–dimensional.
(N) N–dimensional com N>3.

2. Espécie de designação
(T) Todos os objetos maiores (bins) e uma seleção de pedaços menores (ítens)
(S) Uma seleção de objetos maiores e todos os pedaços menores

3. Classificação de objetos maiores
(U) Um objeto
(I) Formatos idênticos
(D) Formatos diferentes

4. Classificação de pequenos ítens
(P) Poucos ítens (de formatos diferentes)
(M) Muitos ítens de muitos formatos diferentes
(R) Muitos ítens de relativamentes poucos formatos diferentes (não–congruentes)
(C) Formatos congruentes

onde **formatos congruentes** são figuras geométricas de mesma forma e tamanho diferindo no máximo com respeito a orientação (e posição), somente no caso multidimensional.

Combinando estes tipos de caracteristicas obtem–se 96 diferentes tipos de problemas C&P abreviadamente denotados pela quadrupla $\alpha/\beta/\gamma/\delta$. Qualquer ausência de símbolo para uma característica significa que qualquer uma das possibilidades é possível.

TABELA 1

Alguns problemas de C&P e tipos combinados correspondentes

Nome	Pertence ao tipo
Knapsack Clássico	1/T/U/
Bin Packing Clássico	1/S/I/M
Cutting Stock Clássico	1/S/I/R
Pallet (plataforma) loading	2/T/U/C
Knapsack multi–dimensional	/T/U/
Dual Bin–Packing	1/T/I/M
Vehicle loading	1/S/I/P, ou 1/S/I/M
Container loading	3/S/I, ou 3/T/U/
Bin Packing bi–dimensional	2/S/D/M
Cutting Stock bi–dimensional (comum)	2/S/I/R
Cutting Stock Geral ou Trim Loss	1,2,3///
Assembly line balancing	1/S/I/M
Multiprocessor scheduling	1/S/I/M
Memory allocation	1/S/I/M

Como torna–se aparente pela notação empregada, os problemas clássicos bin–packing, cutting–stock e vehicle loading pertencem ao mesmo tipo combinado considerando as 3 primeiras características. O que muda essencialmente entre eles é a classificação dos ítens. Considera–se, p. ex., CHVATAL [9, p. 198], que no problema de cutting–stock a lista de demanda possui muitos ítens, porém relativamente poucos deles têm tamanhos diferentes. Para o problema **bin–packing**, ao contrário, considera–se que existam muitos ítens a serem manuseados, onde muitos deles possuem diferentes tamanhos. Em **vehicle loading** somente poucos ítens são considerados. Mas, são conceitos relativos sem um limite definindo quando usar um ou outro termo para o mesmo problema.

Por outro lado, como citamos no início, line balancing, multiprocessor scheduling e memory allocation pertencem ao mesmo tipo combinado como o clássico bin–packing. A diferença entre eles refere–se a características outras além destas básicas. Por exemplo, em **line balancing**, além destas características existe uma ordem de precedência para os ítens.

Embora os termos tenham sido usados indistintamente na literatura, o que nos parece mais apropriado para distingüir **bin–packing** (ou **cutting–stock**) de **loading** (ou múltiplas–mochilas) é que no primeiro *todos* os ítens devem ser empacotados (ou cortados) objetivando minimizar o número de bins (ou pedaços de estoque), enquanto que no segundo *nem todos* os ítens, necessáriamente, precisam ser empacotados e o objetivo é minimizar (ou maximizar) o valor dos ítens carregados.

2.2.2 Formulação Matemática do Problema Clássico

Na literatura, pelo que temos pesquisado, somente quatro formulações do problema bin–packing clássico são apresentadas com algoritmos exatos. Uma está no trabalho de EILON & CHRISTOFIDES ([22], 1971) sob o título de "Loading Problem", a

outra no trabalho de HUNG & BROWN [38] e também no de MARTELLO & TOTH [58], a quarta no trabalho de LEWIS & PARKER [54] para um **modelo matemático generalizado** de bin packing.

2.2.2.1 Formulação de Eilon e Christofides ([22], 1971).

Neste trabalho foram propostas várias formulações do problema bin–packing, algumas com restrições, e também algumas generalizações, assim como formulações de outros problemas relacionados. Alguns objetivos e situações, mostrados naquele trabalho, consideram que existem vetores do tipo (a_i,u_i), onde a_i e u_i $(i=1,...,n)$ são, respectivamente, as magnitudes e os valores dos ítens ; e do tipo (C_j,v_j), onde C_j e v_j $(j=1,2,...,n)$ são, respectivamente, as capacidades e os valores das caixas.

Objetivos:

$[O_1]$ **Objetivo baseado em caixas**

(a) Minimizar o número de caixas usadas;

(b) Minimizar o valor das caixas usadas;

(c) Minimizar o espaço não usado (ou seu valor) das caixas parcialmente usadas.

$[O_2]$ **Objetivo baseado em** ítens

Minimizar o número (ou valor) dos ítens não acomodados nas caixas

$[O_3]$ **Objetivo Combinado**

Minimizar conjuntamente o valor das caixas escolhidas e valor dos ítens não acomodados.

Situações:

(S_1) $\sum_j C_j \geq \sum_i a_i$ e todos os ítens podem ser acomodados nas n caixas;

(S_2) **ou** $\sum_j C_j \geq \sum_i a_i$ mas nem todos os ítens podem ser acomodados, **ou** $\sum_j C_j < \sum_i a_i$

Destas considerações, a seguinte matriz de problemas pode ser formada (veja Fig. II.5):

	0_1			0_2	0_3
	a	b	c		
S_1	1	2	3	–	–
S_2	–	–	4	5	6

FIG. II.5: *Matriz de Problemas de C&P*

Os Problemas:

(1) é o bin–packing clássico com capacidades diferentes;

(2) é uma variante do Problema 1;

(3) é o Problema 1 (ou 2), com imposição de folga mínima;

(4) é um problema de loading, com imposição de folga mínima;

(5) é o problema de múltiplos–knapsacks

(6) é um problema mixto de loading e múltiplos–knapsacks;

(a) Carregamento (loading) de Caixas de mesma Capacidade

Entretanto, naquele trabalho, somente a formulação sobre o problema clássico ($C_j = C$), apresentada a seguir (no caso, com *preprocessamento de* ítens) foi resolvida por um procedimento exato:

$$\text{(BPC1)} \qquad \textit{Min } z \quad = \sum_{j=1}^{\ell} P_j \sum_{i=1}^{n} x_{ij}$$

s.a.

$$\sum_{i=1}^{n} s_i x_{ij} \leq C_j \qquad j = 1,\ldots,\ell \qquad \text{(II.1)}$$

$$\sum_{j=1}^{\ell} x_{ij} = 1 \qquad i = 1,\ldots,n \qquad \text{(II.2)}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \qquad \text{(II.3)}$$

onde: ℓ *limite no número de bins para não sobrar* ítens *(p. ex., ℓ=n, como no texto original, ou algum valor encontrado por um algoritmo aproximativo primal).*

P_j *peso atribuido aos bins, tais que $P_{j+1} > P_j > 0$ (p. ex., $P_{j+1} = \alpha.P_j$ sendo $P_j = 1$ e α o menor número de ítens que pode ser empacotados em um bin).*

s_i *é o tamanho dos* ítens *i.*

C_j *capacidade do bin j.*

x_{ij} *variável binária igual a 1 se o* ítem *i é designado ao bin j, e 0 no caso contrário.*

As restrições de knapsacks (II.1), expressam que o conteúdo total de todos os ítens i (i=1,...,n) designados para um bin j (j=1,...,ℓ) não deve exceder a capacidade comum C dos bins. As restrições de múltipla escolha (II.2), assegura que cada ítem é designado a um e somente um bin, e além disso não deve sobrar nenhum ítem sem ser empacotado; e (II.3) são as restrições de integralidade.

É interessante notar nesta formulação a atribuição de pesos em cada bin, de maneira que quanto maior o índice do bin maior o peso. Isto elimina a necessidade de usar outras variáveis para indicar o uso de um bin, pois, fica assegurado que o menor número de bins será usado. O procedimento exato apresentado para resolver este problema é uma variação do algoritmo de enumeração implícita de Balas [5].

Outras técnicas de solução:

(i) *Relaxação Lagrangeana*

A formulação de (BPC1) se assemelha com a do GAP – GENERALIZED ASSIGMENT PROBLEM (cf. MINOUX [62]), com a particularidade que os pesos dados aos bins indexados estão em ordem decrescentes de valor e suas capacidades de

armazenagem são todas iguais (ou seja, $C_j = C \ \forall j$).

Isto sugere que uma outra maneira de resolver este modelo é através de Relaxação Lagrangeana sugerida por GEOFFRION [31], que passaremos a aplicar teoricamente. A relaxação Lagrangeana forte para o GAP é obtida (cf. GUIGNARD & ROSENWEIN [34]) relaxando as restrições (II.2). O problema Lagrangeano é como segue:

$$(\mathrm{PR}_\lambda) \qquad Min\ w = \sum_{j=1}^{\ell}\sum_{i=1}^{n}(P_j - \lambda_i)x_{ij} + \sum_{i=1}^{n}\lambda_i$$

s.a.

$$\sum_{i=1}^{n} s_i x_{ij} \le C \qquad j = 1,\ldots,\ell$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j$$

Dado um valor para cada λ_i, (PR_λ) se decompõe numa coleção de ℓ problemas de mochila 0–1, um para cada bin j. Isto é:

$$(PR^j): \qquad \underset{1\le j\le \ell}{Max}\ w'_j = \sum_{i=1}^{n}(\bar{\lambda}_i - P_j)x_{ij} \quad \leftarrow \ \ell \text{ – problemas tipo Mochila 0–1}$$

s.a.

$$\sum_{i=1}^{n} s_i x_{ij} \le C$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j$$

O valor da função objetivo do problema Lagrangeano para este valor de $\bar{\lambda}$ é:

$$v(\mathrm{PR}_{\bar{\lambda}}) = - \sum_{j=1}^{\ell} v(\mathrm{PR}^j) + \sum_{i=1}^{n}\bar{\lambda}_i$$

Para achar o λ ótimo, é necessário resolver o problema dual Lagrangeano, o que pode ser feito através de algum método de subgradiente. O dual Lagrangeano é dado por:

$$\textbf{Dual Lagrangeano: } \underset{\lambda \text{ irrestrito}}{\text{Max}} \{ \text{Min } w \} .$$

Usualmente, para resolver problemas deste tipo, trabalha–se com o método de subgradiente de HELD, WOLFE & CROWDER [35].

O valor da solução ótima do problema Lagrangeano fornece um limite melhor, no valor da função objetivo quando usado no método branch–and–bound (em cada nó de enumeração), do que o problema primal contínuo conforme mostra a relação abaixo:

$$z_{\text{rel. linear}} \leq v(PR_\lambda *) \leq z^* \leq v(\text{heurística})$$

A figura a seguir ilustra o funcionamento do método subgradiente:

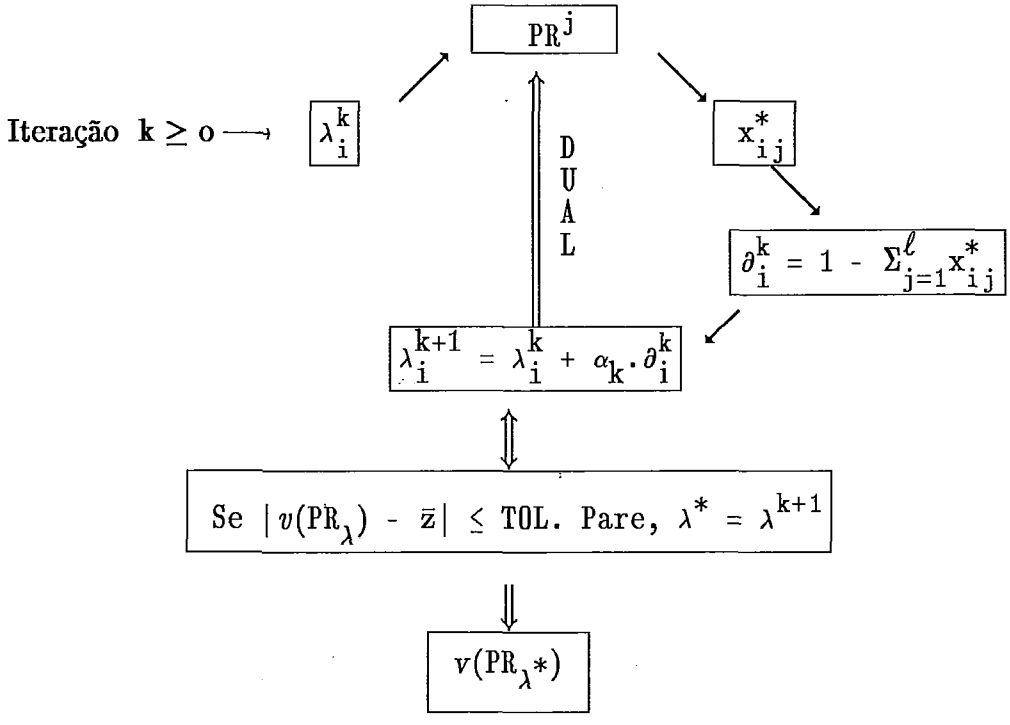


FIG. II.6: *Resolução do Dual Lagrangeano*

O que o esquema acima diz é o seguinte: Na iteração $k \geq 0$, defina o vetor subgradiente (∂_i^k) como

$$\partial_i^k = 1 - \sum_{j=1}^{\ell} x_{ij}^* ,$$

onde x_{ij}^* é a solução de PR^j baseada em λ_i^k. Então, o próximo vetor λ é, para cada componente

$$\lambda_i^{k+1} = \max \{\lambda_i^k + \alpha_k \cdot \partial_i^k, 0\} ,$$

onde α_k é o passo determinado da mesma forma como em [35] e ∂_i^k é uma direção do subgradiente de $v(PR_\lambda)$ em $\lambda = \bar{\lambda}$. Uma vez λ^{k+1} tenha sido calculado voltamos a calcular os ℓ problemas da mochila. O procedimento gera limites na função objetivo que se alternam (i.e., não são melhorados monotonicamente) e frequentemente termina quando atinge um número de iterações arbitrário ou alguma tolerância é satisfeita (p. ex., **dif** $= |v(PR_{\lambda^{k+1}}) - \bar{z}| \leq$ TOL , onde TOL é um pseudo–salto de dualidade máximo admissível); ou, o que é o ideal, se $\| \partial^k \| = 0$.

Recentemente, GUIGNARD & ROSENWEIN [34] sugeriram que a resolução do dual Lagrangeano do GAP fosse feita por um **algoritmo dual ascendente Lagrangeano** ao invés de otimização por subgradiente, pois, em cada iteração deste método há uma melhoria monotônica no limite obtido i.e., $v(PR_\lambda)$ aumenta. Esta técnica possibilita ajustar somente alguns poucos multiplicadores ou variáveis duais em cada nó da enumeração se implantada dentro de um esquema branch–and–bound e, portanto, traz uma vantagem computacional, tanto em termos de memória como em melhor desempenho no processamento. O número de passos ascendentes por nó de enumeração é significativamente menor do que o número de iterações subgradientes requeridas para resolver um idêntico dual Lagrangeano. Funciona basicamente, da seguinte maneira: Se uma solução Lagrangeana viola uma restrição necessária para viabilidade primal, então o

multiplicador Lagrangeano associado com àquela restrição é ajustado no problema dual, resultando num melhoramento do limite Lagrangeano. Os multiplicadores são ajustados pela regra:

$$\lambda^{k+1} = \lambda^{k} + \alpha_{k} \cdot d^{k},$$

onde d^{k} é uma direção de subida avaliada pela troca de multiplicadores correspondentes as restrições violadas. Como efeito do ajustamento há um aumento em $v(PR_{\lambda})$ através do termo

$$\sum_{i=1}^{n} \lambda_{i}[1 - \sum_{j=1}^{\ell} x_{ij}].$$

Desde que a solução x fique inalterada, $v(PR_{\lambda})$ continua a ser aumentado pelo ajuste dos multiplicadores, sendo um problema específico a determinação de um ajuste ótimo. O problema Lagrangeano (PR_{λ}) é resolvido para determinar o valor de $\Delta\lambda_{s}$, para alguma restrição s violada que resulte numa mudança da solução ótima x. Na verdade, os passos ascendentes criam soluções ótimas alternativas. Após término de subida na direção

$$1 - \sum_{j=1}^{\ell} x_{sj},$$

nova iteração é iniciada. Quando não há mais direções ascendentes o processo termina mesmo que uma solução ótima dual não tenha sido encontrada.

(ii) *Decomposição Lagrangeana*

Se as restrições $x = y$ e $y_{ij} = 0$ ou 1, $\forall i,j$, são adicionadas a (BPC1) e as variáveis nas restrições $\sum_{j=1}^{\ell} x_{ij} = 1$ são renomeadas de x para y, a decomposição Lagrangeana para um multiplicador Lagrangeano μ arbitrário, é obtida e dada por:

$$(\mathrm{DRL}_{\mu}) \qquad Min\ w = \sum_{j=1}^{\ell}\sum_{i=1}^{n}(P_j - \mu_{ij})x_{ij} + \sum_{j=1}^{\ell}\sum_{i=1}^{n}\mu_{ij}y_{ij}$$

s.a.

$$\sum_{i=1}^{n} s_i x_{ij} \leq C_j \qquad j = 1,\ldots,\ell$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j$$

$$\sum_{j=1}^{\ell} y_{ij} = 1 \qquad i = 1,\ldots,n$$

$$y_{ij} \in \{0,1\} \qquad \forall i,j$$

(BPC1) se decompõe nos problemas:

$$(\mathrm{D1}) \qquad Min\ w = \sum_{j=1}^{\ell}\sum_{i=1}^{n}(P_j - \mu_{ij})x_{ij}$$

s.a.

$$\sum_{i=1}^{n} s_i x_{ij} \leq C_j \qquad j = 1,\ldots,\ell$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j$$

e

$$(D2) \qquad Min\ w = \sum_{j=1}^{\ell}\sum_{i=1}^{n} \mu_{ij} y_{ij}$$

s.a.

$$\sum_{j=1}^{\ell} y_{ij} = 1 \qquad i = 1,\ldots,n$$

$$y_{ij} \in \{0,1\} \qquad \forall i,j$$

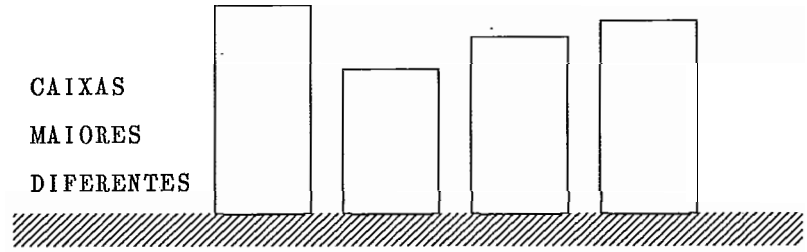
O dual Lagrangeano, neste caso, é:

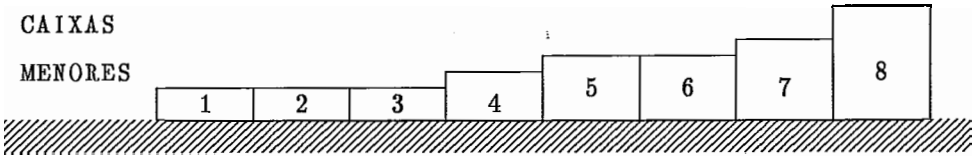
$$\underset{\mu}{Max}\ v(\mathrm{DRL}_{\mu}) = \underset{\mu}{Max}\ v(\mathrm{P1}) + \underset{\mu}{Max}\ v(\mathrm{P2})$$

Entretanto, não há nenhuma vantagem em se trabalhar com este problema dual, porque os problemas envolvidos não são fáceis de serem resolvidos.

(b) Carregamento (loading) para Caixas de Capacidades Diferentes

É dado um número grande N de bins (caixas grandes) de diferentes capacidades e um conjunto de n ítens (caixas pequenas), onde apenas uma dimensão as diferencia. O que se deseja é determinar o menor número de caixas grandes que absorva todos as n caixas pequenas, ou seja, atender o objetivo O_1(a).





A formulação matemática deste problema foi apresentada como segue:

$$
\begin{aligned}
(\text{BPC}2) \qquad & \textit{Min } z = \sum_{j=1}^{N} y_j \\
& \textit{s.a.} \\
& \sum_{i=1}^{n} s_j x_{ij} \leq C_j && j = 1,\ldots,N \\
& \sum_{j=1}^{N} x_{ij} = 1 && i = 1,\ldots,n \\
& \sum_{i=1}^{n} x_{ij} \leq ß y_j && j = 1,\ldots,N \qquad \text{(II.4)} \\
& y_j \,,\, x_{ij} \in \{0,1\} && \forall i,j
\end{aligned}
$$

onde: $y_j = \begin{cases} 1 \text{ se a caixa } j \text{ é carregada com qualquer item,} \\ 0 \text{ se a caixa } j \text{ está completamente vazia.} \end{cases}$

$ß \quad >> n.$

A restrição (II.4) garante que nenhum ítem seja alocado numa caixa cujo $y_j = 0$.

A introdução das variáveis y_j deve–se ao fato que as caixas agora não podem ser penalizadas como antes e o limite superior $\ell \neq N$ para o número de caixas não pode ser considerado.

O problema de bin–packing para este caso, como proposto por FRIESEN & LANGSTON [27], considera ainda a possibilidade de se atribuir ao bin custos proporcionais ao seu tamanho, o objetivo corresponde a minimizar o custo total de empacotamento.

2.2.2.2. Formulação de Hung e Brown ([38], 1978).

É uma modificação da formulação anterior do problema clássico admitindo–se ainda que os bins possam ter ou não diferentes capacidades C_j $(j = 1,...,\ell)$. A formulação é apresentada como segue:

$$(\text{BPC}2) \qquad Min\ z = \sum_{j=1}^{\ell} y_j$$

$$s.a.$$

$$\sum_{i=1}^{n} s_i x_{ij} \leq C_j y_j \qquad j = 1,...,\ell$$

$$\sum_{j=1}^{\ell} x_{ij} = 1 \qquad i = 1,...,n$$

$$y_j\ ,\ x_{ij} \in \{0,1\} \qquad \forall i,j$$

Quanto ao algoritmo desenvolvido e método de solução usa basicamente *enumeração implícita* juntamente com uma caracterização de matrizes de alocação e várias regras que evitam redundancias nas alocações. É feita uma análise comparativa em termos de tempo de solução com a heurística apresentada por Eilon e Christofides em [22] para bins de mesma capacidade e também para capacidades diferentes uniformemente distribuidas em um intervalo previamente estabelecido.

Utilizou–se ainda, para melhorar a eficiência do algoritmo, a *Relaxação Lagrangeana*. O problema dual Lagrangeano foi estabelecido como:

$$\underset{\lambda}{\text{Max}}\ \underset{x}{\text{Min}} \quad \sum_{j=1}^{\ell} y_j + \sum_{i=1}^{n} \lambda_i [1 - \sum_{j=1}^{\ell} x_{ij}]$$

s.a.

$$\sum_{i=1}^{n} s_j x_{ij} \leq C_j y_j \qquad j = 1,\ldots,\ell$$

$$x_{ij} \in \{0,1\}\ ,\ y_j \in \{0,1\} \qquad \forall i,j$$

Assim, dado um valor para cada λ_i , (PR_λ) se decompõe numa coleção de L problemas de mochila 0–1, um para cada bin j. Isto é, para cada j, tem–se que resolver:

$$(\text{PR}^j): \qquad \text{Max w'} = \sum_{i=1}^{n} \bar{\lambda}_i x_{ij} \quad \longleftarrow \begin{array}{l}\ell \text{ - problemas tipo} \\ \text{Mochila 0–1}\end{array}$$

s.a.

$$\sum_{i=1}^{n} s_j x_{ij} \leq C_j \qquad j = 1,\ldots,\ell$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j$$

Tomando–se x^*_{ij} e $\gamma_j = \sum_{i=1}^{n} \bar{\lambda}_i x^*_{ij}$, como a solução ótima e o valor ótimo em (PR^j), respectivamente. Obteve–se, para cada j:

$y_j = 1$ e $x_{ij} = x^*_{ij}$ para i = 1,...,n, se $\gamma_j > 1$, .

$y_j = 1$ e $x_{ij} = 0$ $\forall i$, caso contrário.

Para encontrar λ^*, resolveu–se o problema dual Lagrangeano pelo método de HELD, WOLFE & CROWDER [35]. Porém, a experiência computacional obtida naquele trabalho não recomenda a utilização desta técnica neste tipo de formulação do problema.

Recentemente MARTELO & TOTH ([58],1990) publicaram um livro que traz o assunto de bin–packing como um capítulo específico (capítulo 8), inclusive fornecendo um software desenvolvido em FORTRAN de um algoritmo branch–and–bound por eles desenvolvido, para a resolução desta formulação matemática exposta em HUNG & BROWN ([38],1978). O algoritmo, denominado MTP, faz uso na estratégia de **branching**, do algoritmo "first–fit decreasing".

2.2.3 Formulação de Problemas Relacionados Uni–dimensionais [Variantes].

Em 1975, INGARGIOLA & KORSH [40], apresentaram um algoritmo enumerativo para resolver o problema relacionado de loading com múltiplas mochilas 0–1, o qual admitia a possibilidade de sobra de ítens na solução ótima. A formulação do problema considerado aparece como:

$$
\begin{aligned}
&\textit{Min } z = \Sigma_{j=1}^{\ell}\Sigma_{i=1}^{n} p_i\, x_{ij} \\
&\text{s.a.:} \\
&\qquad \Sigma_{j=1}^{\ell} s_j x_{ij} \leq C_j \qquad\qquad i = 1,\ldots,n \\
&\qquad \Sigma_{i=1}^{n} x_{ij} \leq 1 \qquad\qquad j = 1,\ldots,\ell \qquad\qquad \text{(II.5)} \\
&\qquad x_{ij} \in \{0,1\} \qquad\qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}
\end{aligned}
$$

onde: p_i *é o custo de alocação de um ítem* i.

A restrição (II.5) diz que cada ítem s_j pode ser alocado a um único bin, mas pode haver ítens que sejam rejeitados (i.e., não seja alocado a nenhum bin).

O algoritmo considerado faz, antes de uma busca ser tentada, uma avaliação de todas as possibilidades de inclusão ou exclusão de ítens nas mochilas. A partir daí, estabelece–se uma relação de ordem entre os ítens a qual gera aqueles superfluos que serão excluidos em uma solução ótima se um dado ítem é incluído na mochila.

Fazendo $\ell = 1$, o problema reduz–se ao bem conhecido "Problema com um único knapsack 0-1":

$$Min\ z = \Sigma_{i=1}^{n} p_i y_i$$

$$\text{s.a:} \quad \Sigma_{i=1}^{n} s_i y_i \leq C$$

$$y_i \in \{0,1\} \qquad i = 1,\ldots,n$$

Este problema (e portanto a variante acima) é NP–árduo [28]; isto é, o tempo requerido para uma solução exata pode ser exponencial no tamanho da entrada. O problema da Mochila 0–1 é da ordem de $O(\ell.2^{\ell})$.

Em 1978, HUNG & FISK [39] apresentaram um algoritmo branch–and–bound para resolver o mesmo problema de INGARGIOLA & KORSH. O esquema usou duas alternativas: Relaxação Lagrangeana e Relaxação Surrogate. Na relaxação Lagrangeana o segundo conjunto das restrições são relaxados pelo uso dos multiplicadores duais ótimos calculados no problema linear contínuo, sendo o problema resultante dividido em problemas da mochila para cada bin. Na relaxação substituta ("surrogate") o primeiro conjunto das restrições de mochila é combinado dentro de uma única restrição, e o segundo conjunto de restrições é substituido dentro desta restrição, reduzindo o problema inicial a um problema com uma única mochila.

Em seguida, 1980, MARTELLO & TOTH [59] desenvolveram um esquema branch–and–bound para a solução do problema de Múltiplas Mochilas 0–1, formulado como:

$$Max\ z = \Sigma_{j=1}^{\ell} \Sigma_{i=1}^{n} l_i\ x_{ij}$$

s.a.:

$$\Sigma_{j=1}^{\ell} s_j x_{ij} \leq C_j \qquad i = 1,\ldots,n$$

$$\Sigma_{i=1}^{n} x_{ij} \leq 1 \qquad j = 1,\ldots,\ell$$

$$x_{ij} \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

Supondo que um lucro $l_i = 1$ seja dado para cada a_j e que $C_j = C$ para todo j, a função objetivo quando todas as alocações forem feitas será esgotar ao máximo a lista de ítens a serem empacotados, usando o menor número de bins possível. Por isso, este modelo representa para nós uma variante para o problema de Bin–Packing Clássico.

A Fig. II.7 dá uma ilustração desta formulação e mostra uma solução aproximada com sobra de um ítem e uma solução ótima.

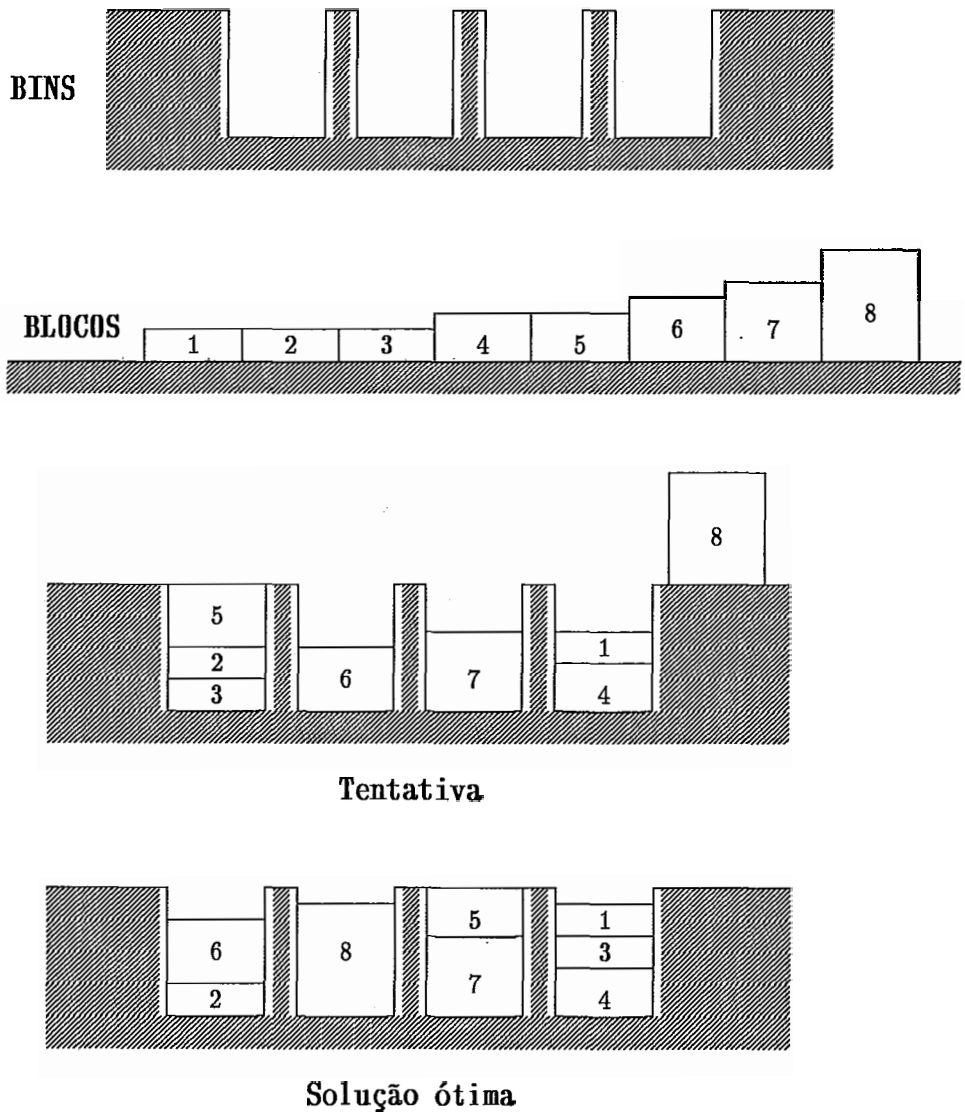


FIG. II.7: *Um exemplo de packing*

O esquema de MARTELLO & TOTH origina vários algoritmos pela utilização de diferentes estratégias de branching e procedimentos de bounding. Os resultados computacionais obtidos, em seu trabalho, foram comparados aos obtidos por HUNG & FISK. Em 1985, MARTELLO & TOTH [60] propuseram um algoritmo mais aperfeiçoado com melhor desempenho do que qualquer outro método para achar soluções exatas até aquela data. Sugeriram que este algoritmo poderia ser usado interrompendo–se a execução após

um número especificado de iterações e utilizando a melhor solução até aquele ponto como uma heurística. No entanto, o objetivo principal naquele modelo geral é maximizar o lucro e não esgotar ao máximo a lista de ítens a serem empacotados. Porém, este algoritmo pode ser empregado (i.e., tomando–se $l_j = 1 \ \forall j$) para resolver esta variante do problema bin–packing clássico. Inclusive, extendida a hipótese quando os bins têm capacidades diferentes.

2.2.4. O Problema Dual.

Existem diversos artigos publicados considerando como o problema Dual de Bin Packing (DBP) a variante do modelo clássico onde são dados m bins de capacidade unitária e $n>m$ ítens com tamanhos em (0,1]. O objetivo é maximizar o número de ítens que podem ser empacotados nos m bins. A dualidade está em que maximizar o número de ítens empacotados implica em minimizar o número de ítens não–empacotados.

Algumas aplicações envolvem programar n tarefas dentro de m processadores paralelos idênticos; arquivos de vários comprimentos dentro de um disco rígido com m cilindros de igual capacidade, etc.

A outra versão de *DBP*, que nos parece a mais apropriada, foi primeiro estudada por ASSMANN *et al.* [1]. A versão é a seguinte: Existe uma lista de números reais positivos $L = \{a_1, a_2, \ldots, a_n\}$ e uma constante positiva, T (teto), $T \geq \max a_i$. O objetivo é empacotar os elementos de L dentro de um número máximo de bins de capacidade $C > T$ tal que a soma dos números em qualquer dos bins seja no mínimo T; i.e., tem–se que completar tantos bins quanto possível. Obviamente, espera–se para isto que todos os bins sejam preenchidos o mais próximo possível do seu limiar T. (O nome "dual" origina–se do "clássico" problema Bin Packing, onde os elementos têm que ser empacotados dentro de um número *mínimo* de bins de tal modo que a soma dos elementos em qualquer bin seja ao menos C).

Este problema modela uma variedade de situações encontradas na indústria e

no comércio, desde empacotamentos de pedaços de frutas dentro de latas tal que cada enlatado possua no mínimo seu peso líquido ou outras especificações (veja ilustração na Fig. II.8), até situações complexas como quebra de monopólios dentro de companhias.

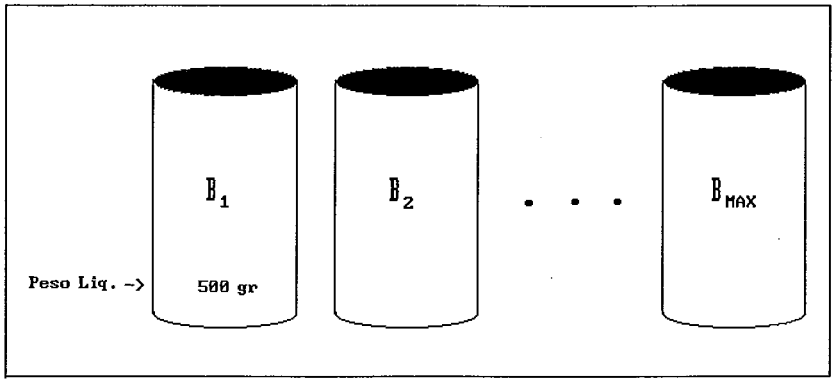


FIG. II.8: *O Problema Dual*

Como no problema clássico, pergunta–se por uma partição de L dentro de um número *máximo* de bins, nenhum dos quais sendo preenchido abaixo de um teto T. Fica clara a relação

$Viabilidade\ Primal \Leftrightarrow Inviabilidade\ Dual$
$Inviabilidade\ Primal \Leftrightarrow Viabilidade\ Dual$

A formulação matemática poderia ser:

$$(DBP) \qquad Max\ z = \sum_{j=1}^{\ell} y_j$$

s.a.

$$\sum_{i=1}^{n} s_j x_{ij} \geq T.y_j \qquad j = 1,\ldots,\ell$$

$$\sum_{j=1}^{\ell} x_{ij} = 1 \qquad i = 1,\ldots,n$$

$$y_j\ ,\ x_{ij} \in \{0,1\} \qquad \forall i,j$$

onde as variáveis são definidas como nos modelos anteriores e $T>0$ é o teto estabelecido para mínimo preenchimento.

2.3 O Bin Packing Generalizado

Em 1980, LEWIS [53] formulou o que seria para o seu estudo "O problema Bin Packing Generalizado":

(BPG) $\quad Min\ Z = \Sigma_{j=1}^{\ell} (p_j y_j + d_j u_j)$

s.a.:

$$\Sigma_{i=1}^{n} s_{ij} x_{ij} + u_j = C_j y_j \qquad j = 1,\ldots,\ell$$

$$\Sigma_{j=1}^{\ell} x_{ij} = 1 \qquad i = 1,\ldots,n$$

$$x_{kj} + x_{hj} \leq 1, \quad (k,h) \in R_j, \qquad 1 \leq j \leq \ell$$

$$y_j\ ,\ x_{ij} \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

onde:

- C_j *é a capacidade do bin j.*
- s_{ij} *é o desgaste do ítem i se designado ao bin j.*
- u_j *é a capacidade não usada do bin j.*
- p_j *é um custo de uso do bin j.*
- d_j *é o custo da capacidade não usada do bin j.*
- R_j *conjunto de restrições de empacotamento a serem obedecidas para quaisquer par de ítens dentro de um mesmo bin j.*

As variáveis binárias x_{ij} e y_j, relacionam–se a designação do ítem i ao bin j e o uso do bin j, respectivamente. Neste caso, quando $p_j = 1$, $d_j = 0$, $C_j = C$ para $j = 1,\ldots,\ell$ e $s_{ij} = s_i$ para todo $i = 1,\ldots,n$, $j = 1,\ldots,\ell$, e $R_j = \emptyset$ para todo j, o problema se reduz ao Bin–Packing Clássico. Da mesma forma, isto acontece se tomarmos $p_j = 0$ e $d_j = 1$ para $j = 1,\ldots,\ell$, pois minimizamos a capacidade não utilizada dos bins no empacotamento

global. Logo, bin packing clássico é um caso especial deste modelo generalizado.

Mais recentemente (1982), LEWIS & PARKER [54] consideraram $R_j = \emptyset$ para todo j. E apresentaram várias metodologias de resolução a este modelo generalizado, mais modesto:

(BPG1)

$$Min\ Z = \Sigma_{j=1}^{\ell} (p_j y_j + d_j u_j)$$

s.a.:

$$\Sigma_{i=1}^{n} s_{ij} x_{ij} + u_j = C_j y_j \qquad j = 1,...,\ell$$

$$\Sigma_{j=1}^{\ell} x_{ij} = 1 \qquad i = 1,...,n$$

$$y_j\ ,\ x_{ij} \in \{0,\ 1\} \qquad \begin{cases} i = 1,...,n \\ j = 1,...,\ell \end{cases}$$

Um exemplo de utilização deste modelo aparece na determinação de **schedule para processadores paralelos**, i.e., a alocação de tarefas a um conjunto de processadores paralelos para minimizar o tempo de término (denominado *makespan*) de todas as tarefas. Admitindo–se um limite inferior de tempo para que todas as tarefas estejam concluídas, o problema é encontrar um empacotamento das tarefas dentro de l bins visando minizar o tempo de processamento das tarefas como um todo, onde cada bin tem capacidade não maior do que o tempo de processamento do limite estipulado. Quando os processadores não são idênticos, o peso do ítem em cada bin que ele satisfaz reflete a interpretação desejada. O custo total de cada bin, corresponde ao preço de utilização do processador e o custo do tempo que o processador fica ocioso quando um dos processadores já tem completado sua designação. Surge, portanto, a penalidade pela capacidade não utilizada. O uso de penalização, contudo, traz uma dificuldade adicional, onde uma solução usando bins mais caros pode ter um custo final menor do que uma solução usando bins mais baratos.

Embora este modelo seja uma generalização de Bin Packing, o tradicional para o problema é minimizar o número de bins, independente de se atribuir pesos aos bins ou pesos aos ítens para diferentes alocações aos bins. Pois assim são considerados, na literatura (ver COFFMAN *et al.* [14]), a maioria dos algoritmos aproximativos sobre este problema.

Apresentamos a seguir, de maneira mais simples, uma parte daquele trabalho:

Fazendo–se a substituição de variáveis $z_j = 1 - y_j$, podemos reconstruir (BPG1) a partir de (BPG2), onde u_j no primeiro conjunto de restrições de (BPG1) é uma variável de folga. O modelo de partida é, então, como segue:

(BPG2): $\quad \text{Min } T \equiv \Sigma_{j=1}^{\ell} (p_j + d_j C_j) -$

$$\text{Max:} \quad \Sigma_{j=1}^{\ell} (p_j + d_j C_j) z_j + \Sigma_{j=1}^{\ell} \Sigma_{i=1}^{n} d_j s_{ij} x_{ij}$$

s .a.:

$$\Sigma_{i=1}^{n} s_{ij} x_{ij} + C_j z_j \leq C_j \qquad j = 1,\ldots,\ell$$

$$\Sigma_{j=1}^{\ell} x_{ij} = 1 \qquad i = 1,\ldots,n$$

$$x_{ij} \, , \, z_j \in \{0, 1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

2.3.1 Programação Linear de Grande Porte

Ignorando as restrições de alocação, a relaxação Lagrangeana de (BPG2) para um dado vetor λ, irrestrito em sinal, é:

(RL): $\quad \Sigma_{j=1}^{\ell} (p_j + d_j C_j) - \Sigma_{i=1}^{n} \lambda_i -$

$$\text{Max:} \quad \Sigma_{j=1}^{\ell} (p_j + d_j C_j) z_j + \Sigma_{j=1}^{\ell} \Sigma_{i=1}^{n} (d_j s_{ij} - \lambda_i) x_{ij}$$

s .a.:

$$\Sigma_{i=1}^{n} s_{ij} x_{ij} + C_j z_j \leq C_j \qquad j = 1,\ldots,\ell$$

$$x_{ij} \, , \, z_j \in \{0, 1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

O problema dual lagrangeano é resolvido determinando–se os multiplicadores duais λ_i, $1 \leq i \leq n$, os quais satisfazem:

(DL1) Min: $-\Sigma_{j=1}^{\ell}(p_j + d_j C_j) + \Sigma_{i=1}^{n}\lambda_i +$

Max $\Big\{ \Sigma_{j=1}^{\ell}(p_j + d_j C_j)z_j + \Sigma_{j=1}^{\ell}\Sigma_{i=1}^{n}(d_j s_{ij} - \lambda_j)x_{ij} \mid$

$\Sigma_{i=1}^{n} s_{ij}x_{ij} + C_j z_j \leq C_j$, $\quad j = 1,...,\ell$,

x_{ij} , $z_j \in \{0,1\}$ $\quad j = 1,...,\ell$ e $i = 1,...,n \Big\}$

s.a: λ_i *irrestrito* $\forall i$.

Porém, se z^*, x^* é a solução ótima de (DL1) para algum dado λ, então

$$(p_j + d_j C_j)z_j^* + \Sigma_{i=1}^{n}(d_j s_{ij} - \lambda_j)x_{ij}^* \geq (p_j + d_j C_j), \quad 1 \leq j \leq l. \tag{II.6}$$

Com efeito, suponha o contrário, i.e., que esta inequação (II.6) seja violada para algum bin j. Então, $z_j^* = 0$ e $x_{ij}^* = 1$ para um ou mais ítens designados a este bin. Mas, o valor desta solução pode ser aumentado fazendo $z_j^* = 1$ e $x_{ij}^* = 0$ para todo ítem i no bin j, os quais contradizem a suposição de otimalidade, causando uma situação absurda.

Temos assim que,

$$(p_j + d_j C_j)z_j^* + \Sigma_{i=1}^{n}(d_j s_{ij} - \lambda_j)x_{ij}^* - (p_j + d_j C_j) \geq 0, \quad 1 \leq j \leq l. \tag{II.7}$$

Quando o bin j está ativo, (II.7) reduz–se a:

$$\Sigma_{i=1}^{n}(d_j s_{ij} - \lambda_j)x_{ij}^* - (p_j + d_j C_j) \geq 0, \quad 1 \leq j \leq l$$

Isto permite reescrever (DL1) como:

(DL2) Min: $\Sigma_{i=1}^{n}\lambda_i + \Sigma_{j=1}^{\ell} q_j$

s.a.:

$$\sum_{i \in I_{jk}} \lambda_i + q_j \geq \sum_{i \in I_{jk}} d_j s_{ij} - (p_j + d_j C_j) \qquad 1 \leq k \leq N_j$$

$$1 \leq j \leq l$$

$q_j \geq 0$, $1 \leq j \leq l$; λ_i *irrestrito* $\forall i$,

onde I_{jk} é o conjunto de ítens no k–ésimo empacotamento viável para o bin j (o que engloba a restrição de empacotamentos de DL1) e N_j é o número de empacotamentos

viáveis para o bin j. (Deve ser notado que foi adicionado e subtraído em DL1 o termo $(p_j + d_j C_j)$, o que possibilita o cancelamento de igual termo em DL1).
Observe que estamos agora com um problema de programação linear de grande porte à variáveis contínuas.

2.3.2 O Programa Mestre

O Dual_PL deste problema é:

$$\text{(DPL)} \quad \text{Min:} \quad \Sigma_{jk} \left[p_j + \underbrace{(d_j C_j - \sum_{i \in I_{jk}} d_j s_{ij})}_{\substack{\text{custo da capacidade} \\ \text{não usada do bin j no} \\ \text{k-ésimo packing viável}}} \right] v_{jk}$$

s.a.:

$$\sum_{jk \in I_i} v_{jk} = 1 \qquad 1 \leq i \leq n \;\rightarrow\; \text{(associada a } \lambda_i \textit{ irrestrito)}$$

$$\sum_{k} v_{jk} \leq 1 \qquad 1 \leq j \leq l$$

$$v_{jk} \geq 0\,, \qquad 1 \leq k \leq N_j,\ 1 \leq j \leq l.$$

onde v_{jk} é o multiplicador dual associado com cada restrição de (DL2) e $I_i = \{(j,k)/$ ítem $i \in I_{jk}\}$. Cada v_{jk} pode ser pensado como uma variável que seleciona um packing I_{jk} ou alguma parte dele, assim Dual_PL é o problema de selecionar um conjunto de empacotamentos, para os bins, tal que o custo seja minimizado, cada ítem é empacotado uma vez apenas, e no máximo um empacotamento é selecionado para cada bin. O custo de um empacotamento é o custo do bin j, p_j mais o custo de qualquer capacidade não usada no bin j.

Resumindo, a relaxação das restrições de designação levou a uma relaxação Lagrangeana para o qual a determinação do valor ótimo equivale a resolver um problema de Programação Linear de Grande Porte nas variáveis duais. Este desenvolvimento analítico é clássico e, segue dos resultados de FISHER, NORTHUP & SHAPIRO [25].

Este problema Dual_PL considerado tem a forma habitual de apresentação:

$$(\bar{D}) \qquad \begin{cases} \text{Min} \sum_{k=1}^{|S|} u_k . f(x^k) \qquad s.a: \\ \sum_{k=1}^{|S|} u_k . g_i(x^k) \leq b \qquad (i=1,\ldots,m) \\ \sum_{k=1}^{|S|} u_k = 1 \ , \qquad u_k \geq 0 \quad (k=1,\ldots,|S|) \end{cases}$$

onde $f(x^k)$ é um função real, $g(x^k)$ é uma função de $\mathbb{R}^n$ em $\mathbb{R}^1$, S é um conjunto de elementos discretos, e x^k é um elemento no conjunto S das soluções viáveis de todos os subproblemas de mochila referentes a cada bin. O problema Dual_PL compreende uma combinação linear convexa dos vértices de cada submodelo das restições de empacotamento para cada bin j. Esta formulação, por sua vez, possui I+1 linhas e o número de variáveis ("colunas") quantas forem os elementos de S. Obviamente, na prática, trabalha–se com um problema mestre restrito e usa–se a técnica de "geração de colunas" para criar colunas para entrar na base quando elas são necessárias. Para ver como isto é feito ver LASDON [52], também MACULAN, CAMPELLO & RODRIGUES [56]. Um detalhe que deve ser notado é que se a variável u_k for inteira 0-1 este problema é semelhante ao problema inicial (BPG2).

O problema Dual_PL foi reduzido por uma restrição que limita o número de empacotamentos que poderiam ser selecionados, através da limitação no número de bins requerido (resultado fácil e rapidamente obtido por um método heurístico aproximativo quase–ótimo). O problema mestre restrito é o seguinte:

(DPL1) Min: $\Sigma_{jk}\left[p_j + (d_j C_j - \sum_{i \in I_{jk}} d_j s_{ij})\right] v_{jk}$

s.a.:

$$\sum_{jk \in I_i} v_{jk} = 1 \qquad 1 \leq i \leq n$$

$$\sum_{k} v_{jk} \leq 1 \qquad 1 \leq j \leq l$$

$$v_{jk} \geq l, \text{ e } v_{jk} \geq 0 \qquad 1 \leq k \leq N_j, \; 1 \leq j \leq l.$$

onde

l *é limite inferior no número de bins requerido.*

2.3.3 O Procedimento de Geração de Colunas

1) Introduza uma variável de folga para cada bin. Determine um valor para *LI*. Se uma solução viável de (BPG2) é conhecida, gere os empacotamentos para esta solução e adicione essas colunas ao problema.

2) Resolva o programa mestre restrito obtendo um conjunto de multiplicadores duais, correspondentes as suas restrições. Seja λ^* o vetor de multiplicadores correspondente ao conjunto das primeiras restrições.

3) Para cada bin j $(j = 1,...,\ell)$, resolva os problemas satélites, i.e., os problemas da mochila do tipo:

Max: $\Sigma_{j=1}^{\ell} (d_j s_{ij} + \lambda_j^*) x_{ij}$

s.a.: $\Sigma_{i=1}^{n} s_{ij} x_{ij} \leq C_j$

3) Para cada bin j, teste se a coluna gerada, pela solução ótima do passo (2), é uma coluna que melhora o valor do problema restrito. Em caso positivo esta coluna é introduzida ao problema.

4) Se ao menos uma coluna gerada produz uma melhoria, vá ao passo 1. Caso contrário, a solução atual do problema mestre restrito é ótima. Se a solução atual contém variáveis artificiais, então não existe solução viável para o problema primal (BPG2).

2.3.4 O Gap de Dualidade

Desde que o problema de Programação Linear (DPL) resulta de uma relaxação Lagrangeana das restrições de designação, pode ser que que o valor de sua solução ótima seja menor do que o valor da solução ótima de (BPG2) produzindo uma folga ou GAP de dualidade. Mesmo que não exista o gap pode acontecer que a solução de Dual_PL seja fracionária. Uma forma de contornar esta diferença que possa existir entre o valor da solução primal e o valor da solução dual é através do método Branch–and–Bound.

2.3.5 O Esquema Branch–and–Bound

O problema Primal de partida, no qual será usado relaxação Lagrangeana, foi derivado de BPG1, fazendo–se $u_j = w_j y_j - \Sigma_{i=1}^{n} s_{ij} x_{ij}$. O problema resultante é:

(BPG3): $\quad \text{Min } z \equiv \Sigma_{j=1}^{\ell} (p_j + d_j C_j) - \Sigma_{j=1}^{\ell} \Sigma_{i=1}^{n} d_j s_{ij} x_{ij}$

s.a.:

$$\Sigma_{i=1}^{n} s_{ij} x_{ij} \leq C_j \qquad j = 1,\ldots,\ell$$

$$\Sigma_{j=1}^{\ell} x_{ij} = 1 \qquad i = 1,\ldots,n$$

$$x_{ij}\ ,\ z_j \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

A ralaxação Lagrangeana deste novo problema para um dado λ é:

(RLBB) $\Sigma_{i=1}^{n}\lambda_i$ + Min: $\Sigma_{j=1}^{\ell}(p_j + d_jC_j)y_j - \Sigma_{j=1}^{\ell}\Sigma_{i=1}^{n}(d_js_{ij} - \lambda_i)x_{ij}$

s.a.:

$$\Sigma_{i=1}^{n} s_{ij}x_{ij} \leq C_j y_j \qquad j = 1,\ldots,\ell$$

$$x_{ij}\ ,\ z_j \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

Como, nesta formulação, não há restrições no número de bins que são requeridos para resolver o problema, a relaxação foi aumentada com a introdução da seguinte restrição de capacidades dos bins:

$$\Sigma_{j=1}^{\ell}\ \overline{a}_j y_j \geq n$$

onde $\overline{a}_j$ é o número máximo de objetos que podem ser empacotados no bin j. Qualquer solução viável deve permitir que n ítens sejam empacotados. O problema, então, torna–se:

(RLBB1) $\Sigma_{i=1}^{n}\lambda_i$ + Min: $\Sigma_{j=1}^{\ell}(p_j + d_jC_j)y_j - \Sigma_{j=1}^{\ell}\Sigma_{i=1}^{n}(d_js_{ij} - \lambda_i)x_{ij}$

s.a.:

$$\Sigma_{i=1}^{n} s_{ij}x_{ij} \leq C_j y_j \qquad j = 1,\ldots,\ell$$

$$\Sigma_{j=1}^{\ell}\ \overline{a}_j y_j \geq n$$

$$x_{ij}\ ,\ z_j \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

O procedimento para solução deste problema consiste de:

(1) Resolver para cada bin e um dado λ, o seguinte problema de knapsack:

Maximizar: $\Sigma_{j=1}^{\ell}(d_js_{ij} - \lambda_i)x_{ij}$

s.a.:

$$\Sigma_{i=1}^{n} s_{ij}x_{ij} \leq C_j$$

$$x_{ij} \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,n \\ j = 1,\ldots,\ell \end{cases}$$

Seja x* a solução ótima deste problema para cada bin j e seja

$$v_j = p_j + d_j C_j - \Sigma_{i=1}^{n}(d_j s_{ij} - \lambda_i)x^*_{ij}$$

(2) Para cada bin j resolver:

$$\Sigma_{i=1}^{n} x_{ij} = \bar{a}_j$$

s.a.:

$$\Sigma_{i=1}^{n} s_{ij} x_{ij} \leq C_j$$

$$x_{ij} \quad \text{binário}$$

Sejam x_j' a solução ótima e $\bar{a}_j = \Sigma_{i=1}^{n} x_{ij}'$.

(3) Resolver o seguinte problema de knapsack:

Min: $\Sigma_{j=1}^{\ell} v_j y_j$

s.a.:

$$\Sigma_{j=1}^{\ell} \bar{a}_j y_j \geq n$$

$$y_j \in \{0,1\} \qquad j = 1,\ldots,\ell$$

Sejam y^* a solução ótima para este problema e $v(RL) = \Sigma_{i=1}^{n}\lambda_i + \Sigma_{j=1}^{\ell} v_j y_j^*$ o valor da relaxação Lagrangeana.

O procedimento *Branch–and–Bound* para resolução exata de (BPG2) é sugerido como segue:

Passo (0): Resolver o (DPL1). Se a solução for inteira, pare. Faça a lista de subproblemas vazia, e p = 0. Seja B^* um bound canditado de valor positivo grande. Execute os Passos de 1 a 4 e depois vá ao Passo 5.

Passo (1): Usando os multiplicadores Lagrangeanos do Passo 0, resolva (RLBB1) encontrando o bound corrente e a correspondente solução (x,y). Se $B \geq B^*$ vá ao Passo 4.

Passo (2): Se para a corrente solução $\Sigma_{j\in J} x_{ij} \neq 1 \ \forall i$, onde $J = (j/y_j = 1\}$, então vá ao Passo 3.

Atualize o bound candidato e a solução (x^*,y^*) com o corrente bound e solução correspondente, e remova todos os subproblemas, da lista de subproblemas, com um bound igual ou maior do que o bound candidato. Vá ao Passo 4.

Passo (3): Sejam $I_1 = \left\{ i \ / \ \Sigma_{j\in J} x_{ij} \geq 1 \right\}$ e $I_2 = \left\{ i \ / \ \Sigma_{j\in J} x_{ij} = 0 \right\}$.

Se $I_1 = \emptyset$, vá ao Passo 3.1. Faça $J_i = \{j \ / \ j\in J, \ i\in I_1, \text{ e } x_{ij} = 1 \ \}$.

Selecione para <u>branching</u> as variável x_{kl} tal que

$$\theta_{kl} = \max_{i \in I_1} \left\{ \max_{j \in J_i} \{ (d_j s_{ij} + \lambda_i^*) / s_{ij} \} \right\}.$$

Adicione o subproblema a lista de subproblema e vá para 4.

(3.1) Sejam $J_i' = \{ j \ / \ j\in J$ e o objeto $i\in I_2$ pode ser alocado ao bin $j\}$. Se $J_i' = \emptyset \ \forall i\in I_2$, vá para o Passo 3.2. Selecione para <u>branching</u> a variável x_{kl} tal que

$$\theta_{kl} = \max_{i \in I_2} \left\{ \max_{j \in J_i'} \{ (d_j s_{ij} + \lambda_i^*) / s_{ij} \} \right\}.$$

Adicione o subproblema a lista de subproblemas e vá ao Passo 4.

(3.2) Sejam $J_i'' = \{ j \ / \ j\notin J$ e o objeto $i\in I_2$ pode ser alocado ao bin $j\}$. Se $J_i'' = \emptyset \ \forall i$, uma solução viável não existe, vá para o Passo 4. Selecione para <u>branching</u> a variável x_{kl} tal que

$$\theta_{kl} = \min_{i \in I_2} \left\{ \min_{j \in J_i''} \{ v_j / \bar{a}_j \} \right\} .$$

Adicione o subproblema a lista de subproblema e vá ao Passo 4.

Passo (4): Faça $p \leftarrow p + 1$.

Passo (5): Selecione da lista de subproblemas o subproblema com o menor bound inferior. Se a lista está vazia, PARE.

Passo (6): Seja $x_{kl} = 0$ e execute o Passo de 1 até 4. Faça $x_{kl} = 1$ e $y_l = 1$, e execute Passos de 1 a 4. Vá ao Passo 5.

Resumo: Para resolver (BPG2) pelo esquema B–B exposto, primeiro determina–se os multiplicadores duais ótimos através de um procedimento de geração de colunas. Em seguida, o esquema B–B é chamado para encontrar a solução ótima primal.

Capítulo III
O Problema Bin–Packing : Algoritmos Aproximativos

3.1 Algumas definições

Esta seção faz uma rápida explanação da terminologia usada em estudos de problemas combinatoriais, definindo–se o que significam termos como "algoritmos aproximativos" ou "comportamento do pior–caso".

3.1.1 Algoritmo Aproximativo

Um algoritmo A para um problema de otimização $P=(E_P,S_P,m_P)$ é dito um *algoritmo aproximativo* se, dado qualquer instância $I \in E_P$, A encontra uma candidata a solução $\sigma \in S_P(I)$. E denota–se $A(I) = m_P(I,\sigma)$. Se $A(I) = OPT(I)$ para todo $I \in E_P$, então A é dito um *algoritmo exato*.

Na literatura sobre Bin–Packing, heurísticas rápidas (i.e., de tempo polinomial) que produzem soluções viáveis próximas do ótimo são referidas, quase sempre, como *algoritmos aproximativos*. Doravante estaremos levando em conta este conceito.

Uma solução viável aproximada de BIN PACKING é qualquer empacotamento dos ítens dados dentro dos bins, o qual não excede a capacidade dos bins. Uma solução dual viável é, ao contrário, uma solução que ultrapassa a capacidade ou um limite pré–estabelecido do bin. Consequentemente, tal empacotamento fornece menor ou igual quantidade de bins do que a solução ótima primal.

3.1.2 Garantia de Performance de Algoritmos Aproximativos

Basicamente, uma garantia de performance para um algoritmo aproximativo é estabelecida por um teorema que traz embutido uma função que limita o comportamento do algoritmo no pior–caso, para qualquer instância. Para o problema de Bin–Packing o teorema é descrito pelo enunciado: "Para todas as instâncias I, $A(I) \leq r \cdot OPT(I) + d$ ", onde $r \geq 1$ e d são constantes especificadas. Em geral, a constante d é assintoticamente desprezada e o fator dominante é a razão r.

Ao se analisar um algoritmo aproximativo é de interesse determinar a **melhor** garantia possível, i.e., o menor r para o qual um teorema dessa forma pode ser provado. Pode–se mostrar que nenhum limite melhor pode existir construindo instâncias do problema na qual o algoritmo comporta–se tão ruim quanto o limite r. Mais precisamente, se pudermos mostrar que para alguma constante d' e todo $N>0$ existe instâncias I do problema com $OPT(I) > N$ e $A(I) > r'.OPT(I) - d'$, então nenhum limite de performance pode ser provado com $r < r'$. Diz–se que temos determinado o comportamento do pior–caso de um algoritmo quando a razão r da garantia for igual a razão r' das instâncias exemplo. Este valor comum é a menor razão que pode ser garantida pelo algoritmo, e é denominada de *razão de performance do pior–caso* ou simplesmente *razão de performance* para o algoritmo.

Para um problema de maximização, uma garantia de performance pode ser colocada na forma $OPT(I) \leq r.A(I) + d$. $A(I)$ e $OPT(I)$ são trocados para que o domínio de valores possíveis de r fique $1 \leq r \leq \infty$. Com isso, as razões de performance para ambos problemas de minimização e maximização podem ser vistas dentro de uma mesma escala de valores; e portanto a performance de algoritmos aproximativos para ambos problemas podem ser comparadas.

3.1.3 Medidas de performance de Algoritmos Aproximativos

Algoritmos heurísticos para o problema de bin packing são geralmente

avaliados através de sua performance no pior–caso. Sejam P um problema de minimização (ou maximização), e I uma instância qualquer de P, a razão $R_A(I)$, definida por

$$R_A(I) = \frac{A(I)}{OPT(I)}$$

e

$$\left[R_A(I) = \frac{OPT(I)}{A(I)}\text{, no caso de maximização}\right]$$

dá uma medida de performance do algoritmo aproximativo A, i.e., expressa, de uma maneira razoável, a proximidade de sua solução à otimalidade. Assim, as seguintes medidas de performance **formalizam** a análise do pior–caso:

(1) *razão de performance absoluta*

$$R_A = inf\{r \geq 1:\ R_A(I) \leq r \ \ para\ toda\ instância\ I\}$$

(2) *razão de performance assintótica*

$$R_A^{\infty} = inf\{r \geq 1:\ para\ algum\ N>0,\ R_A(I) \leq r \ \ para\ todo\ \ I\ \ com\ \ OPT(I) \geq N\}$$

A *melhor performance assintótica atingível* para um problema de otimização P é dada por:

$$R_{MIN}(P) = inf\{r \geq 1:\ existe\ um\ algoritmo\ aproximativo\ A,\ para\ P,\ com\ \ R_A^{\infty} = r\}$$

Neste caso, A é o melhor algoritmo aproximativo possível para P.

3.1.4 Esquema de Aproximação

Dado um problema de otimização combinatória P, um *esquema de aproximação* é um algoritmo aproximativo A tal que se dado um requerimento de exatidão $\epsilon>0$ (i.e., uma exigência de performance mínima) e uma instância $I \in E_P$, A **deriva** um algoritmo aproximativo A_ϵ tal que

$$R_{A_\epsilon}(I) \leq 1 + \epsilon.$$

O termo esquema de aproximação é usado porque para cada ϵ é derivado um algoritmo A_ϵ. Em outras palavras, um esquema de aproximação é um conjunto de tais algoritmos $\{A_\epsilon : \epsilon>0\}$.

Entretanto, sucessivamente melhores aproximações são obtidas a custo de maior complexidade de tempo. Deseja–se, naturalmente, que o tempo requerido pelos algoritmos sucessivos seja polinomial no tamanho do problema e não cresça tão rapidamente quando ϵ decresce. **Daí porque impor que o tempo seja polinomial em $1/\epsilon$.**

Um esquema de aproximação é dito *polinomial* se para cada $\epsilon>0$, o algoritmo derivado A_ϵ é de complexidade de tempo polinomial no tamanho de I. Porém, se A_ϵ tem complexidade de tempo polinomial no tamanho de I e de $\frac{1}{\epsilon}$, então A é dito um *esquema de aproximação de tempo totalmente polinomial.*

Em Fernandes de La Vega & Lueker [23] é apresentado um **esquema de aproximação** para o problema bin–packing clássico, o qual para qualquer $\epsilon>0$, produz um algoritmo aproximativo A_ϵ com razão de performance $1+\epsilon$. Mais precisamente, a garantia provada é que $A_\epsilon(I) \leq (1+\epsilon)\cdot OPT(I) + O(\epsilon^{-2})$. O esquema tem complexidade de tempo polinomial em n, o número de itens, mas exponencial em $\frac{1}{\epsilon}$.

Karmarkar & Karp [46] extenderam este esquema de aproximação para um

com complexidade de tempo polinomial em n e $\frac{1}{\epsilon}$. Apresentaram, portanto, um esquema de aproximação **totalmente** polinomial e um algoritmo A, *off–line*, de tempo polinomial foi derivado, que garante um empacotamento usando não mais do que $OPT(I) + O(log^2(OPT(I)))$ bins e portanto tem $R_A^{\infty} = 1$. Contudo, este algoritmo além de exigir uma complicada programação tem um complexidade computacional da ordem de $O(n^8 log^3 n))$. O algoritmo usa o método elipsóide de KHACHIAN [48] e GROTSCHEL *et al.* [33] como uma subrotina, como também subrotinas para achar soluções próximas do ótimo para o "problema da mochila" NP–árduo.

Vê–se, portanto, que embora $R_A^{\infty} = 1$ continuamos com a questão $P = NP$?!

Os resultados acima, são interessantes do ponto de vista teórico, mas improváveis de ter qualquer colocação em aplicações práticas de BP, **justificando a nossa pesquisa por novos algoritmos aproximativos de muito menos complexidade computacional do que o esquema acima e comportamento do pior–caso semelhante a *FFD*.**

3.1.5 Algoritmos pseudo–polinomiais

É sabido que a complexidade de um algoritmo é calculada em função do *tamanho da entrada* de sua codificação. No sistema binário um número k é representado por

$$\begin{cases} \lfloor \log_2 k + 1 \rfloor , & k \in \mathbb{Z} \backslash \{0\} \\ 1 \quad , & k = 0 \end{cases}$$

dígitos e esta codificação é usualmente aceita. Um algoritmo cuja complexidade é polinomial ou exponencial levando em conta esta codificação na base 2, não tem sua complexidade alterada quando a entrada é codificada numa base maior. Entretanto, o resultado não é o mesmo, se for utilizada uma representação unária (TOSCANI & SZWARCFITER [69]).

Um algoritmo A é dito de complexidade *pseudo–polinomial*, quando sua complexidade é polinomial para o tamanho da entrada, quando a entrada é codificada no sistema unário.

Um problema NP–completo que admite algoritmo pseudo–polinomial é dito *NP–completo fraco*. Problemas cuja possível existência de algoritmo pseudo–polinomial implica na igualdade $P = NP$, são chamados de *NP–completo forte*. O problema de Bin–Packing é NP–completo forte (cf. GAREY & JOHNSON [28]).

Para um estudo mais completo a respeito de análise do pior–caso de algoritmos aproximativos, ver [20,24,28], em português TOSCANI & SZWARCFITER [69].

3.1.6 Um Exemplo Completo – O Algoritmo NEXT FIT

O problema bin–packing foi um dos primeiros trabalhos que contribuiram para provar que algoritmos aproximativos rápidos poderiam realmente **garantir** soluções próximas do ótimo ou subótimas, para qualquer instância de entrada (veja JOHNSON [40]). Isto é importante porque facilita a decisão se é melhor ou não pagar por *algoritmos exatos* que consomem mais tempo de computação.

Considere o seguinte procedimento heurístico chamado, NEXT FIT: Empacotar os itens de $L = \{a_1, a_2, \ldots, a_n\}$ um de cada vez, iniciando com a_1, o qual é colocado no bin B_1. Suponha agora que a_i seja o próximo a ser empacotado, e seja B_j o bin não vazio de maior índice. Se a_i couber em B_j (a soma dos tamanhos dos itens não excederá $C - s(a_i)$), onde $s(a_i)$ é o tamanho do item a_i e C é a capacidade comum dos bins, então colocar a_i no bin B_j. Caso contrário, iniciar o bin B_{j+1} colocando a_i dentro dele (veja Fig. III.1).

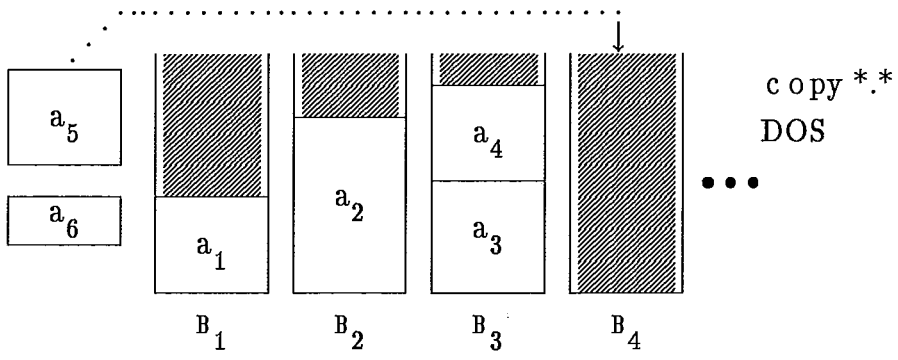


FIG. III.1: *A estratégia Next–Fit.*

Como se pode notar, pela forma como os itens são empacotados, este é um algoritmo muito rápido (de tempo linear).

3.1.5.1 Performance do Algoritmo NEXT–FIT

Teorema 1 (Garantia de Performance). Dado um problema BIN–PACKING, seja *NF(L)* a solução encontrada pelo algoritmo NEXT–FIT e *OPT(L)* a solução ótima. Então,

$$NF(L) \leq 2 . OPT(L)$$

Demonstração: A demonstração é simples. Seja

$c(B_j)$ = conteúdo do *bin* B_j $(j=1,...,m)$.

Para qualquer *bin* B_j $(j = 1,...,m-1)$, temos

$$c(B_j) + c(B_{j+1}) > C$$

senão os objetos de B_{j+1} deveriam ser colocados em B_j. *Sem perda de generalidade*, admite–se que a capacidade de qualquer bin é *1*.

Somando–se **todos** os conteúdos dos bins não vazios, temos

$$c(B_1) + c(B_2) +...+ c(B_{m-1}) + c(B_m) > \frac{m}{2} \quad \text{p/ } m \text{ par}$$

e

$$[c(B_1) + c(B_2) +...+ c(B_{m-2}) + c(B_{m-1})] + c(B_m) > \frac{m-1}{2} \quad \text{p/ } m \text{ ímpar}$$

Como cada bin tem capacidade 1, o empacotamento ótimo necessita de ao menos

$$OPT(L) > \frac{m-1}{2} \quad \Rightarrow \quad m < 2 \cdot OPT(L) + 1$$

ou melhor

$$NF(L) \leq 2.OPT(L) \blacksquare$$

Para saber se $r = 2$ é o menor valor possível para o qual o teorema pode ser provado, seja L a seguinte família de instâncias para P:

$$L = \{\frac{1}{2}, \frac{1}{2N}, \frac{1}{2}, \frac{1}{2N}, \ldots, \frac{1}{2}\} \quad com \quad 4N - 1 \ objetos$$

Isto requer N bins para $2N$ itens de tamanho $1/2$ e todos os itens de tamanho $1/2N$ dentro de um único bin desde que $(2N-1) \times \frac{1}{2N} < 1$.

Neste caso,

$$OPT(L) = N + 1 \text{ bins} \quad \Rightarrow \quad N = OPT(L) - 1$$

e

$$NF(L) = 2N \text{ bins} \quad \Rightarrow \quad NF(L) = 2 \cdot OPT(L) - 2$$

A Fig. III.2 abaixo ilustra o exemplo.

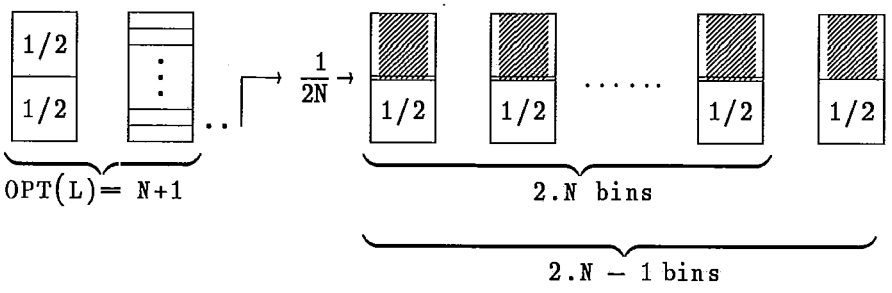


FIG. III.2: *Exemplo de uma lista L com* $NF(L) > 2.OPT(L) - 3$

Portanto, como para alguma constante $d' = 3$ e todo $N>0$ existe instâncias L com $OPT(L) > N$ e $NF(L) > 2 \cdot OPT(L) - 3$, então $r = 2$ é a melhor razão de performance para o pior–caso deste . Pois, para $L \to \infty$, $OPT(L) \to \infty \Rightarrow R^{\infty}_{NF}(L) = 2$. Isto significa 100% de erro, o que não pode ser tolerado. Para melhorar este limite foram criadas outras heurísticas, que serão mostradas na seção 3.2.

3.2 – Algoritmos Aproximativos Básicos Primais

Como os algoritmos exatos para problemas NP–árduo requerem uma pesquisa combinatorial exaustiva, uma alternativa é usar algoritmos heurísticos rápidos que constroem soluções viáveis próximas do ótimo em tempo polinomial. O problema bin–packing como formulado no modelo clássico é um **sistema de independência**, o que significa que admite algoritmos míopes para obtenção de uma solução aproximada próxima do ótimo. D. S. JOHNSON[1] apresentou quatro algoritmos heurísticos míopes básicos, além de NEXT FIT, e provou resultados interessantes com respeito aos limites do pior–caso.

Estes algoritmos são transcritos a seguir juntamente com as medidas de performance no pior–caso:

Heurística First Fit [FF]

Passo 0. Sejam os bins indexados com $B_1, B_2, \ldots$, faça $i \leftarrow 1$.

Passo 1. Ache o menor j tal que B_j está no nível $\beta \leq 1 - a_i$, e coloque a_i em B_j. Se ele não satisfaz em qualquer bin B_j aberto, abra um novo bin.

Passo 2. Faça $i \leftarrow i + 1$. Se $i \leq n$, repita o Passo 1. Caso contrário, pare e retorne o máximo j.

---

[1]DAVID S. JOHNSON é pioneiro no desenvolvimento de algoritmos heurísticos rápidos para o problemas bin–packing. Em 1973, defendeu sua tese de Ph.D. intitulada "Near–optimal bin–packing algorithms" onde apresentou, para este problema, vários algoritmos heurísticos simples com provas de razão de performance bem definidas.

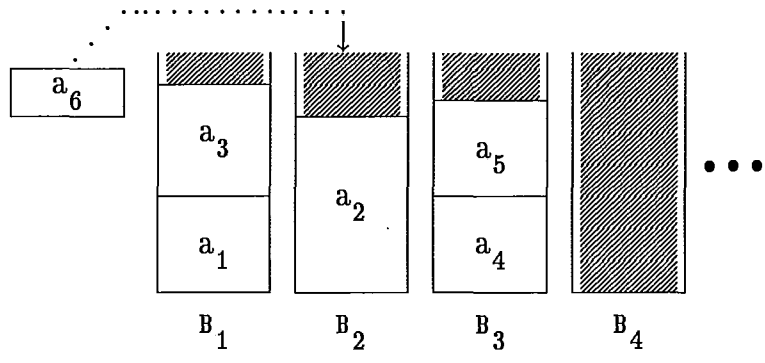


FIG. III.3: *Exemplo de First Fit*

Heurística Best Fit [BF]

Passo 0. Sejam os bins indexados com $B_1, B_2, \ldots$, faça $i \leftarrow 1$.

Passo 1. Ache o menor j tal que B_j está no nível $ß \leq 1 - a_i$, onde ß é o maior possível, e coloque a_i em B_j. Se ele não satisfaz em qualquer bin B_j aberto, abra um novo bin.

Passo 2. Faça $i \leftarrow i + 1$. Se $i \leq n$, repita o Passo 1. Caso contrário, pare e retorne o máximo j.

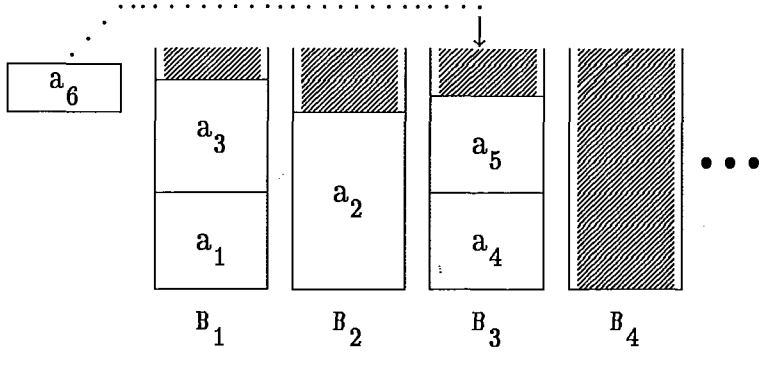


FIG. III.4: *Exemplo de Best Fit*

Heurística First Fit Decreasing [FFD]

Passo 0'. Ordene os ítens por tamanhos não–crescentes e aplique First (Best) Fit à lista obtida.

Heurística Best Fit Decreasing [BFD]

Passo 0'. Ordene os ítens por tamanhos não–crescentes e aplique Best Fit à lista obtida.

Heurística Next Fit Decreasing [NFD]

Passo 0'. Ordene os ítens por tamanhos não–crescentes e aplique Next Fit à lista obtida.

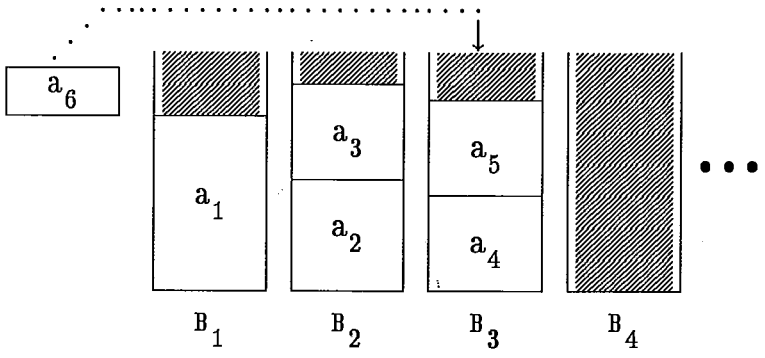


FIG. III.5: *Exemplo de Next Fit Decreasing*

3.2.1 Garantia de Performance (Heurísticas Básicas)

Utilizando–se a notação $FF(L)$, $BF(L)$, $FFD(L)$, $BFD(L)$ e $NFD(L)$ para denotar o número de bins usados ao aplicar–se cada um dos algoritmos acima, respectivamente, para a lista L, as seguintes medidas do pior–caso foram obtidas quando $L \to \infty$:

$$R_{FF}^{\infty} = R_{BF}^{\infty} = \frac{17}{10} = 1.7$$

$$R_{FFD}^{\infty} = R_{BFD}^{\infty} = \frac{11}{9} \cong 1.22$$

$$R_{NFD}^{\infty} \cong 1.69$$

Todos estes limites, assim como ligeiras variações dos mesmos, podem ser construídos, particularmente, para pequenos valores de k (Fig. III.6). De sorte, que resultados mais gerais foram obtidos para quaisquer que sejam as listas L. Estes resultados são expressos através de teoremas, o primeiro dos quais abordaremos com algum detalhe por ter servido de modelo para a garantia de performance de outros algoritmos aproximativos posteriores para este problema. Procuramos melhorar o esclarecimento do uso de funções de peso, com ilustrações gráficas e algumas provas de propriedades das mesmas, pois conforme cita COFFMAN [11] *"embora as funções de peso sejam capazes de verificar a corretividade na prova de limites assintóticos, suas origens, como também a idéia das provas, são envolvidas em mistério..."* . O segundo teorema possui demonstração de uma dificuldade considerável e demanda muitas páginas (originalmente, acima de 100 págs.), por isso esta não será apresentada. Um esboço da prova se encontra em [44]. Para o terceiro, será apenas comentado como foi provada a sua garantia de performance.

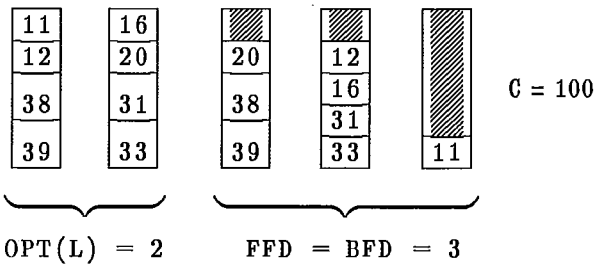


FIG. III.6: *Exemplo de lista L com* $FFD(L)/OPT(L) \cong 1.5$

Teorema 2 [Jonhson, Demers, Ullman, Garey & Graham,1974]

Para qualquer que seja a lista L:

$$\{BF(L)\} \quad FF(L) \leq \frac{17}{10} OPT(L) + 2 \ bins$$

A demonstração do teorema necessita de uma preparação matemática inicial:

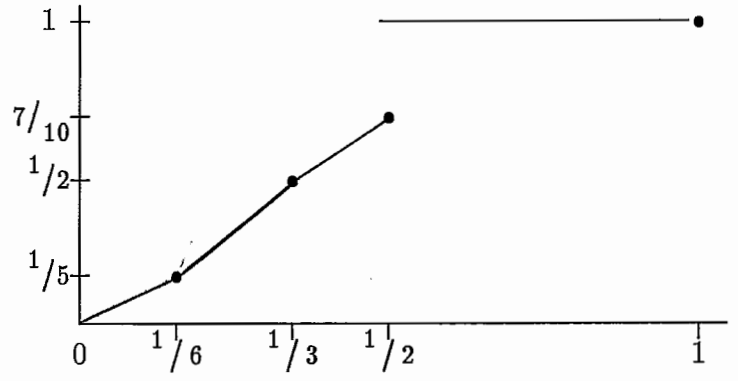


FIG. III.7: *A função* $w(x)$

Seja $w : [0,1] \to [0,1]$ uma função definida como segue (veja Fig. III.7):

$$w(x) = \begin{cases} \frac{6}{5}x & \text{se} \quad 0 \le x \le \frac{1}{6} \\ \frac{9}{5}x - \frac{1}{10} & \text{se} \quad \frac{1}{6} < x \le \frac{1}{3} \\ \frac{6}{5}x + \frac{1}{10} & \text{se} \quad \frac{1}{3} < x \le \frac{1}{2} \\ 1 & \text{se} \quad \frac{1}{2} < x \le 1 \end{cases}$$

Então, a função w possui as seguintes propriedades (transcrito de CHVATAL & ARMSTRONG [10]):

1) Se $1/2 < x \le 1$, então $w(x) = 1$

2) Se $k \ge 2$, $x_1 \ge x_2 \ge \ldots \ge x_k$, $x_1 + x_2 + \ldots + x_k \le 1$
e $w(x_1) + w(x_2) + \ldots + w(x_k) = 1 - s$ para $s > 0$,
então

$$x_1 + x_2 + \ldots + x_k \le 1 - x_k - \frac{5}{9}s$$

3) Se $x_1 + x_2 + \ldots + x_k \le 1$, então

$$w(x_1) + w(x_2) + \ldots + w(x_k) \le \frac{17}{10}.$$

Entenda $w(x_i)$ como o peso do i–ésimo objeto. De forma que o peso do compartimento inteiro será a soma dos pesos dos objetos nele contido.

Prova:

A verificação destas propriedades encontra–se no Apêndice A.

A idéia da construção desta função e o porquê do número mágico $\frac{17}{10}$, no teorema, é mostrada em GAREY *et al.* [29] onde é feita a seguinte **conjectura numérico–teórica**:

Se $L = \{a_1, a_2, \ldots, a_n\}$ *é uma lista ordenada de itens com tamanhos* $s(a_i)$ *obedecendo* $0 < s(a_i) \leq 1$. *Seja* $FF(L)$ *o número de bins de tamanho 1 requerido pelo algoritmo "First Fit" para empacotar L. Sejam*

$$R_{FF}^{\infty}[\alpha] = \lim_{N \to \infty} \sup[(1/N) \cdot \max\{FF(L): L \text{ pode ser empacotado dentro de } N \text{ bins de tamanho } \alpha\}$$

e

$$w(\alpha) = \max\{\omega(\bar{P}): \bar{P} \text{ é uma partição de } \alpha\}.$$

Então,

$$\boxed{R_{FF}^{\infty}[\alpha] = w(\alpha)}$$

onde:

α *é um número racional positivo.*

Uma partição (ou decomposição viável) de α é qualquer sequência $\bar{P} = (P_1, P_2, \ldots)$ de **inteiros** maiores do que *um*, $2 \leq P_1 \leq P_2 \leq \ldots$, dos quais ao menos dois deles $\neq 2$, tal que $\alpha = \sum_{i=1}^{k} \frac{1}{P_i}$ (k é finito).

$w(\bar{P}) = \sum_{i=1}^{k} \frac{1}{(P_i - 1)}$ é o **peso de uma partição** $\bar{P}$ de α.

Para um exemplo, seja $\alpha = 1$. Então, $P_1 = 2$, $P_2 = 3$ e $P_3 = 6$. $P_2 \neq 2$ para não violar a condição de ao menos 2 dos $P_i \neq 2$. O procedimento termina após P_3 ser

escolhido, pois $1 - 1/2 - 1/3 - 1/6 = 0$. Portanto, $\omega(1) = 1 + 1/2 + 1/5 = \frac{17}{10}$. É conhecido de [29] que $R^{\infty}_{FF}[1] = \frac{17}{10}$ também.

Um contra–exemplo para esta conjectura foi apresentado em SHEARER [66]. Entretanto, seu trabalho confirmou, quando α é um racional, que se na expansão de $\alpha = \Sigma_{i=1}^{k} 1/P_i$ encontrando $\omega(\alpha)$ os P_i aumentam suficientemente rápido então $R(\alpha) = \omega(\alpha)$.

Segue da decomposição de α, parte da construção da função $w(x)$ apresentada acima (ver Fig. III.8).

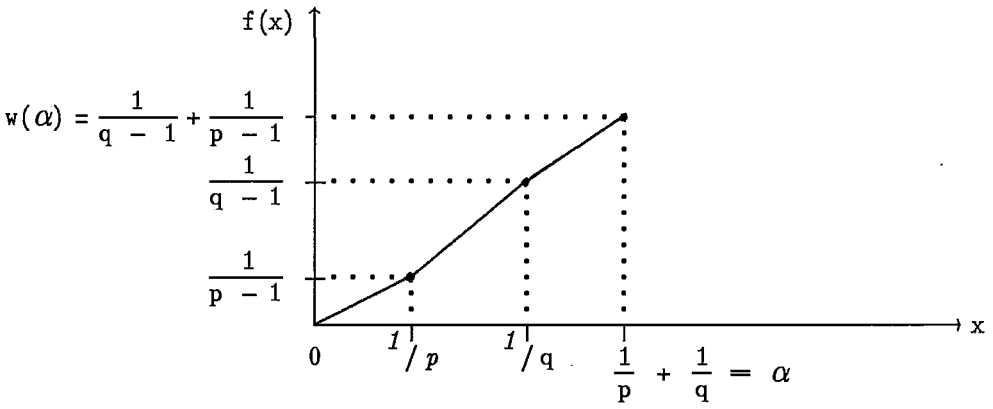


FIG. III.8: *A função penalidade* $f:(0,\alpha] \to (0,w(\alpha)]$

Demonstração do Teorema 2

Sejam $L = (x_1,x_2,\ldots,x_n)$ e $B_1^*, B_2^*,\ldots, B_m^*$ compartimentos cujos empacotamentos por FIRST–FIT (BEST–FIT) tem um peso menor que *1*, digamos $1 - s_1, 1 - s_2,\ldots, 1 - s_m$, respectivamente.

Denotando:

$\overline{c}_i$ = capacidade não utilizada do bin B_i

$\overline{c}_0 = 0$

Então,

$$\overline{c}_i > \frac{5}{9} s_i + \overline{c}_{i-1} \qquad i = 1,\ldots,m-1$$

Com efeito, $x \in B_i^*$ é tal que $x \leq \frac{1}{2}$, pois para $\frac{1}{2} < x \leq 1$, $w(x) = 1$ (pela proposição 1). Logo $\bar{c}_i < \frac{1}{2}$, senão $x \in B_m^*$ teria sido colocado num dos B_i^*, $(i = 1,..., m-1)$ ao invés de B_m^*. Portanto, cada $B_1^*,...,B_{m-1}^*$ deve conter pelo menos dois objetos.

Seja B_i^* com $1 \leq i \leq m-1$ e admitamos que $x_1,...,x_k \in B_i^*$ com $x_1 \geq x_2 \geq ... \geq x_k$ $(k \geq 2)$. Pela propriedade 2,

$$x_1 +...+ x_k \leq 1 - x_k - \frac{5}{9} s_i \therefore x_k + \frac{5}{9} s_i \leq \underbrace{1 - (x_1 +...+ x_k)}_{\bar{c}_i}$$

ou

$$\bar{c}_i \geq \frac{5}{9} s_i + x_k > \frac{5}{9} s_i + \bar{c}_{i-1} .$$

pois $x_k > \bar{c}_i$ (porque senão x_k deveria ter sido colocado em B_{i-1}^*).

Com isto, temos que

$$\Sigma_{i=1}^{m-1} s_i < \frac{9}{5} \cdot \Sigma_{i=1}^{m-1}(\bar{c}_i - \bar{c}_{i-1}) = \frac{9}{5} \bar{c}_{m-1} < \frac{9}{10}$$

e mais ainda

$$\Sigma_{i=1}^{m} s_i < \frac{9}{10} + s_m < 2 .$$

Agora temos:

Para $B_1^* \longrightarrow w(x_1) + ... + w(x_k) = 1 - s_1$

$B_2^* \longrightarrow w(x_{k+1}) + ...+ w(x_r) = 1 - s_2$

até

$B_m^* \longrightarrow$ com peso $1 - s_m$

Para B_{m+1}^* em diante $w(x) = 1$.

Logo,

$$\Sigma_{j=1}^{n} w(x_j) = n - \underbrace{(s_1 + s_2 + \ldots + s_m)}_{\Sigma_{i=1}^{m} s_i < 2} \geq n - 2 \geq FF(L) - 2$$

Por outro lado, aplicando a proposição 3 aos bins de um packing ótimo de L, obtemos

$$\Sigma_{j=1}^{n} w(x_j) \leq \frac{17}{10} OPT(L)$$

Então,

$$FF(L) - 2 \leq \Sigma_{j=1}^{n} w(x_j) \leq \frac{17}{10} OPT(L)$$

ou melhor

$$\boxed{FF(L) \leq \frac{17}{10} OPT(L) + 2 \text{ bins}}$$

concluindo a prova do teorema. ■

Para a comprovação de que $\frac{17}{10}$ é a melhor razão de performance do pior–caso para *FF [BF]*, basta agora mostrarmos uma família de instancias *L* na qual $FF(L) \geq \frac{17}{10} OPT(L) - d'$, onde *d'* é uma constante. Em JONHSON *et al.* [44] é mostrado um exemplo de tais listas, porém elas são de exposição longa para ser mostrada neste trabalho. Dessa forma, a Figura 5 ilustra exemplos mais simples (também apresentado naquele trabalho, mas com uma pequena mudança na quantidade dos elementos em cada região) os quais aproximam–se daquela razão.

EXEMPLO 2: Para qualquer $N \in \mathbb{Z}^+$ e $0 < \delta < \frac{1}{84}$, seja a lista $L = \{a_1, a_2, \ldots, a_n\}$ definida por

$$a_i = \begin{cases} \frac{1}{6} - 2\delta, & 1 \leq i \leq 6N \\ \frac{1}{3} + \delta, & 6N < i \leq 12N \\ \frac{1}{2} + \delta, & 12N < i \leq 18N \end{cases}$$

Claramente $OPT(L) = 6N$, pois cada um dos elementos das três regiões distintas de L, conjuntamente, podem ser empacotados perfeitamente num bin. Portanto, usando–se a regra **first–fit** ou **best–fit** obtêm–se o empacotamento, o qual consiste de:

N bins contendo cada um, 6 elementos de tamanho $\frac{1}{6} - 2\delta$;
$3N$ bins contendo cada um, 2 elementos de tamanho $\frac{1}{3} + \delta$;
$6N$ bins contendo cada um, 1 elemento de tamanho $\frac{1}{2} + \delta$.

Os dois empacotamentos são mostrados na Fig. III.9:

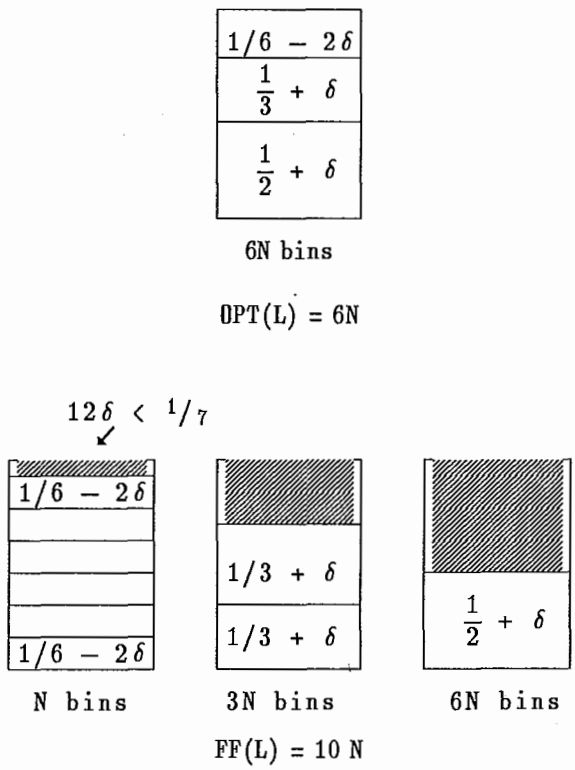


FIG. III.9: *Exemplos de listas L com* $FF(L) = \frac{5}{3} \cdot OPT(L)$.

Portanto, temos

$$\frac{FF(L)}{OPT(L)} = \frac{BF(L)}{OPT(L)} = \frac{N + (3N) + (6N)}{6N} = \frac{5}{3} = 1.666...$$

Teorema 3. [Johnson, 1973]

Para qualquer que seja a lista L:

$$[BFD(L)] \qquad FFD(L) \leq \frac{11}{9} OPT(L) + 4 \text{ bins}$$

Demonstração: A demonstração completa envolve uma análise extremamente detalhada e ocupa mais de 100 páginas [43]. Existe em [44, pp. 308–324] uma **indicação** de como foi realizada a prova deste teorema. ■

O exemplo a seguir mostra uma situação na qual o limite assintótico do teorema 3 é atingido para a regra *FFD*.

Exemplo 3: Sejam $C = 60$ e $L = \{31(\text{x}6), 17(\text{x}6), 16(\text{x}6), 13(\text{x}12)\}$, então (veja Fig. III.10) $FFD(L) = \frac{11}{9} \cdot OPT(L)$.

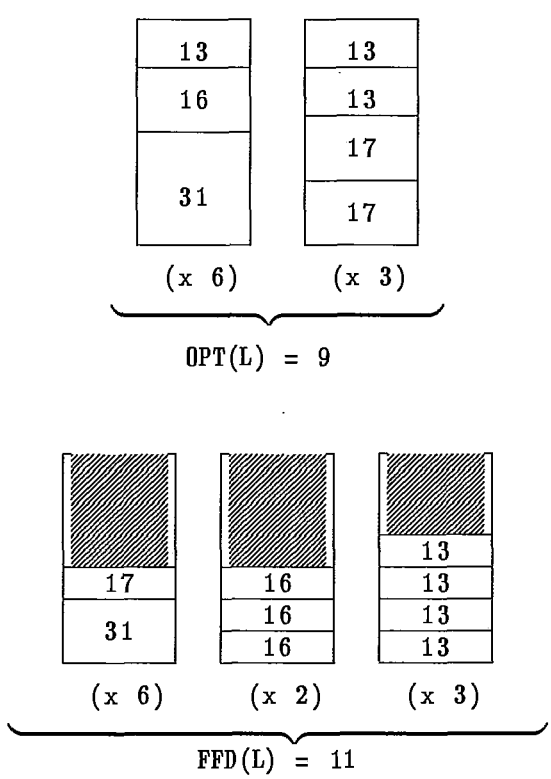


FIG. III.10: *Exemplo de uma lista L com* $\frac{FFD(L)}{OPT(L)} = \frac{11}{9}$.

Uma razão para a demonstração do teorema 2 ser muito longa, e consequentemente de complexidade surpreendente, segundo CHVATAL & ARMSTRONG [10], é que o algoritmo *First Fit*, consequentemente *FFD*, apresenta um comportamento não–intuitivo (veja Figs. III.11 à III.13). Isto é, se por exemplo, L = {7,9,*1*,6,2,4,3} e C = 13, então $FF(L) = 4$. Porém, se o objeto de tamanho *1* é retirado da lista *L*, $FFD(L) = 4$ bins. Por outro, se L = {760,395,395,379,379,241,200,105,105,40} e $C = 1000$, então $FFD(L) = 3$, mas, se os tamanhos dos objetos na lista forem todos diminuídos de uma unidade, temos $FFD(L) = 4$.

ANOMALIAS

FIRST FIT: L = {7,9,*1*,6,2,4,3} e C = 13

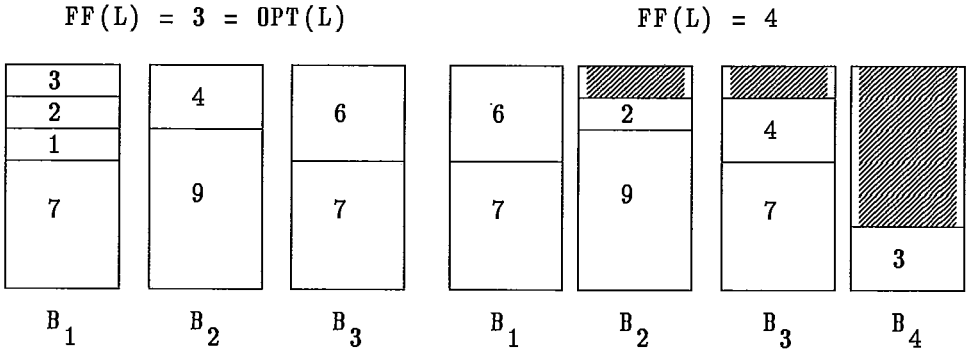


FIG. III.11

FIRST FIT DECREASING

L = {760, 395, 395, 379, 379, 241, 200, 105, 105, 40} e C = 1000

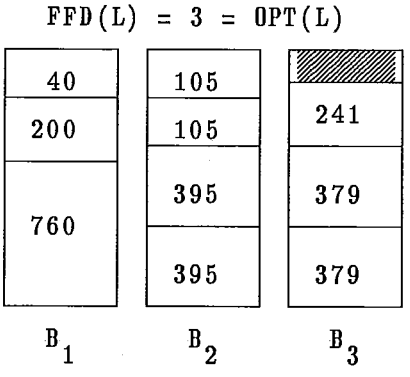


FIG. III.12

L = {759, 394, 394, 378, 378, 240, 199, 104, 104, 39}

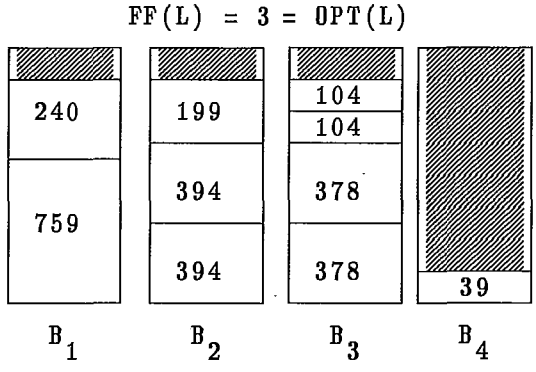


FIG. III.13

3.2 Classes Gerais de algoritmos Heurísticos de Empacotamento

Em JOHNSON [42], as regras de empacotamento FIRST FIT e BEST FIT são mostradas pertencerem a uma classe mais geral de regras de empacotamento todas tendo o mesmo comportamento no pior–caso. Se as listas de entrada estão em ordem decrescente, o comportamento do pior–caso destas regras de empacotamento, na classe, é consideravelmente melhorado e, se não for o mesmo para todas, ao menos é restrita a um campo reduzido de possibilidades. É ainda mostrado que qualquer implementação de uma regra de empacotamento na classe requer ao menos $O(nlogn)$ comparações. São apresentados outros algoritmos de tempo polinomial de baixa ordem (tempo–linear) para obter soluções "próximas do ótimo" para o problema de bin–packing.

Os algoritmos são divididos segundo satisfaçam as seguintes duas restrições:

(1) **Restrição ANY FIT** : Se o bin B_j está vazio, ele não pode ser escolhido a menos que o objeto a_i (próximo da lista a ser designado) não satisfaça em qualquer bin à esquerda de B_j. Isto é, somente se utiliza um novo bin quando um item não satisfaz em *qualquer* bin já aberto.

(2) **Restrição ALMOST ANY FIT** : Se o bin B_j é o único cuja folga é a maior

dentre aquelas de todos os outros bins, então ele não pode ser escolhido a menos que a_i não satisfaça em qualquer bin à esquerda de B_j. Ou seja, evitar sempre que possível colocar ítens em bins de maior folga.

Assim foram denominados de:

AF – o conjunto de todas as regras de empacotamento que obedecem a primeira restrição; e,

AAF – o conjunto, mais restrito, de todas as regras que obedecem ambas as restrições.

Os membros de AF e AAF são chamados de algoritmos ANY FIT e ALMOST ANY FIT, respectivamente. Os algoritmos FIRST FIT e BEST FIT se enquadram dentro da segunda classificação, e portanto são algoritmos ALMOST ANY FIT [$\subseteq$ ANY FIT].

Como medida do pior–caso para um S, usa–se a quantidade

$$R_S^{\infty}(t) = inf\{r \geq 1: \frac{S(L)}{OPT(L)} \leq r, \forall L \text{ tal que } x \in L \text{ e } 0<x\leq t\leq 1\}$$

significando que nenhum número de todas as listas L é maior do que $0 < t \leq 1$.

Esta medida é importante em situações onde o maior ítem esperado é significativamente menor do que a capacidade do bin.

Note–se que se $t_2 \geq t_1$, uma lista sem números maiores do que t_1 é também uma lista sem números maiores do que t_2, e assim $R_S^{\infty}(t)$ é uma **função crescente** em t, com o máximo valor sendo $R_S^{\infty}=R_S^{\infty}(1)$.

Foi mostrado em [44] que, para os dois algoritmos FIRST FIT e BEST FIT e $t > \frac{1}{2}$,

$$R_S^{\infty}(t) = 1.7$$

e, para $m = \lfloor 1/t \rfloor \geq 2$,

$$R_S^{\infty}(t) = 1 + \frac{1}{m}$$

Estes resultados são mostrados em [42] serem extendidos para as classe dos algoritmos ALMOST ANY FIT (AAF). Além disso, se L está em ordem decrescente, mostra–se lá também que resultados como

$$11/9 \leq R_{SD} \leq 5/4$$

valem para uma classe mais geral de algoritmos, os ANY FIT (AF).

Os valores precisos de $R_S^{\infty}(t)$ como uma função de t foram obtidos para muitos dos algoritmos [42,44]. Exceto para os algoritmos *WORST FIT* e *NEXT FIT*, os quais produzem a função contínua $R_S^{\infty}(t) = 1 + \frac{t}{(1-t)}$, esses tendem a ser funções degraus determinadas por $\lfloor 1/t \rfloor$ (veja Fig. III.14).

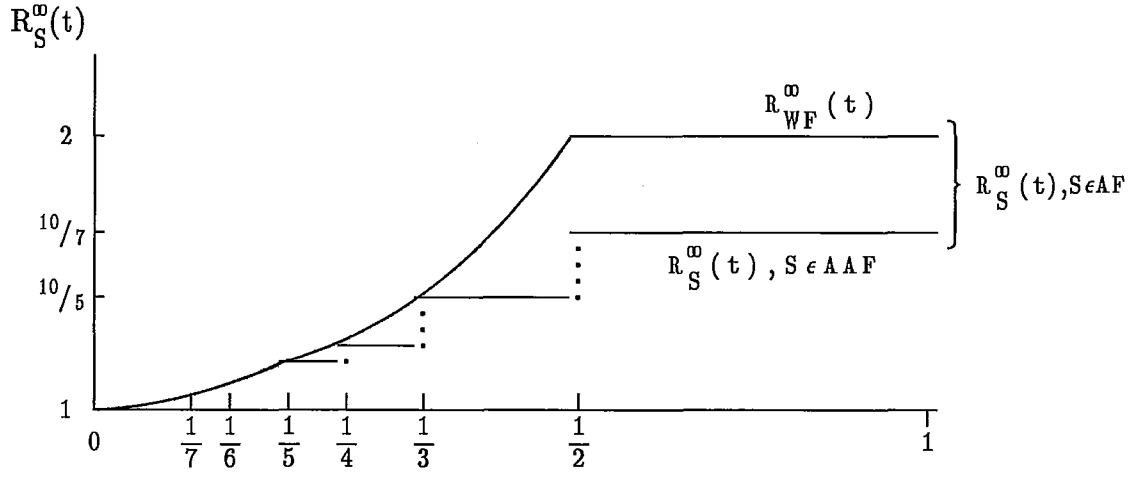


FIG. III.14: $R_S^{\infty}(t)$ *como uma função do algoritmo S e* $t \epsilon (0,1]$.

A Tabela 2 a seguir, realça uma série de resultados que têm sido encontrados para o problema bin–packing clássico (cf. JOHNSON [42]).

TABELA 2

Bounds do pior–caso assintótico para problemas bin–packing

Algoritmo	Complexidade	$R_S^{\infty}(t), \frac{1}{2}<t\leq 1$	$R_S^{\infty}(t), m=\lfloor 1/t \rfloor \geq 2$
1. Worst Fit	O(nlogn)	2.0	$1 + \frac{1}{1/t - 1}$
2. Next Fit	O(n)	2.0	
3. Next k-Fit $k \geq 2$	O(n)	$[1.7+\frac{3}{10k}, 2.0]$	$1 + \frac{1}{m}$
4. Group-X Fit	O(n)	[1.7,2.0]	
5. First Fit	O(nlogn)	1.7	
6. Best Fit	O(nlogn)		
7. Almost Any Fit	O(nlogn)		
8. S $\in$ AAF	$\geq$ O(nlogn)		
9. Group-X Fit Group d	O(n)	1.5	$1 + \frac{1}{m + 1}$
10. S-X Grouped S $\in$ AF	$\geq$ O(nlogn)	$\frac{4}{3} = 1.333...$	
11. FF Decreasing	O(nlogn)	$\frac{11}{9} = 1.222..$	$[\, 1 + \frac{1}{m + 2} - \frac{2}{m(m+1)(m+2)}$, $1 + \frac{1}{m + 2}\,]$
12. BF Decreasing	O(nlogn)		
13. S Decreasing SD $\in$ AF	$\geq$ O(nlogn)	[11/9,5/4]	

A seguir apresentamos as definições dos algoritmos, conhecidos desde 1973 [42,43,44], destacados na Tabela 2:

WORST FIT (WF): coloca a_i no bin não–vazio com a maior sobra (gap), com empate quebrado a favor do bin de índice mais baixo.

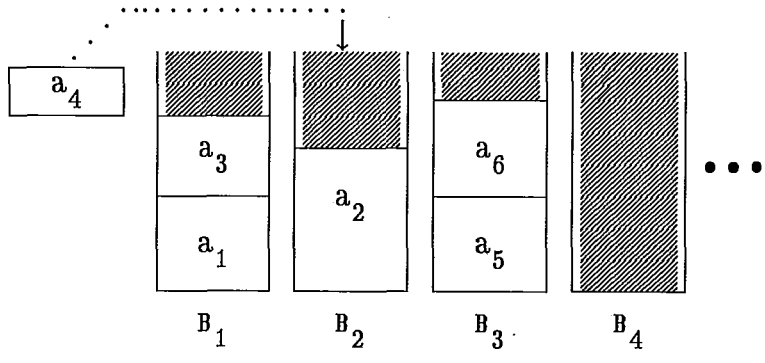


FIG. III.15: *Exemplo de Worst Fit*

NEXT–k FIT, k≥1: é como NEXT FIT exceto que coloca a_i num novo bin somente se ele não satisfaz em qualquer dos últimos k bins não–vazios.

GROUP–*X*–FIT: A idéia básica é substituir as comparações diretas entre objetos e gap de bins ou entre gap de bins e gap de bins por comparações contra algum padrão fixado o qual serve para classificar os objetos e gaps dentro de grupos cujos membros têm tamanhos similares. Para isso, seja uma escala de intervalos um conjunto $X = \{x_1,\ldots,x_k\}$, onde $x_1 = 0$, $x_k = 1$ e $x_i < x_{i+1}$, $1 \leq i \leq k$. Assim X é uma partição do intervalo unitário [0,1] dentro de subintervalos

$$[x_1,x_2),[x_2,x_3),\ldots,[x_{k-1},x_k),\{1\}.$$

$[x_i,x_{i+1})$ é chamado intervalo X_i, com intervalo X_k sendo $\{1\}$.

A regra de empacotamento GROUP–*X*–FIT faz o seguinte: para designar um objeto a, seja $i' = min\{i: x_i \geq s(a)$ *e para algum* j, $1 \leq j \leq n$, $gap(B_j) \in X_i\}$. Escolher B_j tal que $gap(B_j) \in X_i$, sujeito a restrição que se $B_j = \emptyset$, $j = min\{j': nivel(B_{j'}) = 0\}$.

ALMOST ANY FIT (AWF): tenta o segundo maior gap primeiro, e então processa como WORST FIT – isto faz uma diferença (ver JOHNSON [42]).

GROUP–*X*–GROUPED (GXFG): Faz um preprocessamento dos elementos de L de acordo com a regra GROUPING–X, definida logo a seguir, e então empacota os elementos segundo GROUP–X FIT.

A regra GROUPING–X ordena os elementos de L de maneira que para todo $x_i \in X$, se $s(a) > x_i$, $s(b) \leq x_i$ então $pos(a) < pos(b)$ (onde, $pos(a)$ = posição do elemento a em L).

Conclusão: Os resultados para ALMOST ANY FIT são os mesmos como aqueles para ALMOST WORST FIT (todos dos quais obedecem a regra básica *AAF*), enquanto os resultados de ANY FIT são os mesmos como aqueles para WORST FIT, o qual essencialmente faz a pior escolha permitida sob a regra básica *AF*. ANY FIT DECREASING e todos os algoritmos DECREASING obdecem a regra básica ANY FIT tendo em vista que atingem o mesmo bound como *FFD*, embora o melhor que tenha sido provado para qualquer desses (outros do que *FFD* e *BFD*) é que $R^{\infty}_{SD} \leq 5/4 = 1.25$ [42,43].

Uma outra classe de algoritmos que mereceu atenção especial consiste dos algoritmos **on–line**, os quais designam ítens para bins na ordem em que eles chegam para processamento, sem conhecimento a cerca dos ítens subsequentes na lista. FIRST FIT DECREASING, por exemplo, não é um algoritmo on–line, desde que ele primeiro re-ordena a lista. Estes algoritmos são importantes porque eles podem ser os únicos a serem utilizados em certas situações, onde os ítens têm que ser designados para os bins tão logo cheguem. A procura de um melhor limite de performance do pior–caso para um desta classe foi alvo de muitas pesquisas. YAO [70] baseado na análise de exemplos do pior–caso para o FIRST FIT, deduziu um novo, o REVISED FIRST FIT (*RFF*), com

$R^{\infty}_{RFF} = 5/3 \simeq 1.666...$, o qual é melhor quando comparado a $R^{\infty}_{FF} = 1.7$; limite este até o trabalho de YAO o melhor desta classe. Além disso, seu trabalho mostrou que para **qualquer** algoritmo on–line S, devemos ter $R^{\infty}_{S} \geq 1.5$.

3.4 Histórico do melhor limite de performance do pior–caso:

O algoritmo FIRST FIT DECREASING para o problema bin–packing foi por longo tempo, década de 70, o campeão dos algoritmos aproximativos por sua garantia que nenhum empacotamento que ele gera usa mais do que $(11/9)\mathrm{OPT}(L) + 4$ bins. Por muitos anos uma variedade de tentativas para melhorar o limite assintótico de 11/9 foram feitas. Muitos dos progressos alcançados foram teóricos ao contrário de práticos. O primeiro deles foi deduzido em 1978 e publicado em 1980, YAO [70], chamado REFINED FIRST FIT DECREASING (RFFD) o qual roda em tempo $O(n^{10}\log n)$ e proporcionou uma garantia que

$$\mathrm{RFFD(L)} \leq \left[\frac{11}{9} - \frac{1}{10.000.000}\right]\mathrm{OPT}(L) + 8 \text{ bins.}$$

Em seguida (1981), FERNANDEZ DE LA VEGA & LUEKER [23] mostrou, de um ponto de vista teórico, um resultado muito melhor do que este: Para qualquer $\varepsilon>0$, existe um algoritmo A_{ε} de *tempo linear* que tem uma razão de pior–caso assintótica não maior do que $1+\varepsilon$. Mais precisamente, a garantia provada é que $A_{\varepsilon} \leq (1+\varepsilon)\mathrm{OPT}(L) + \mathrm{O}(\varepsilon^{-2})$. Infelizmente o tempo de execução para este esquema de aproximação é exponencial em $1/\varepsilon$. Este obstáculo foi depois superado por KAMARKAR & KARP [46,1982]. Eles deduziram versões modificadas de A_{ε} com tempo de execução crescendo somente polinomialmente com $1/\varepsilon$. Na verdade, o esquema faz uso de técnicas avançadas como o método elipsóide de KHACHIYAN [48,1979] e GROTSCHEL *et al.* [33,1981], assim como subrotinas do "problema de knapsack". Um algoritmo A foi gerado cuja garantia de performance usa não mais do que $\mathrm{OPT}(L) + O(\log^2\mathrm{OPT}(L))$ bins e portanto tem $R^{\infty}_{A} = 1$, mas a complexidade computacional foi aumentada para $O(n^8\log^3 n)$. Esse avanço, entretanto, é

considerado sem efeito em aplicações práticas de bin packing, porque o esquema de aproximação é complicado para programar e caro para usar.

O mais recente algoritmo, com ainda um potencial prático, é uma versão modificada do FIRST FIT DECREASING (*MFFD*) apresentada em JOHNSON & GAREY [45,1985], o qual melhora o comportamento do pior–caso de *FFD*, sem um aumento significativo no tempo de execução e complexidade de programação. A garantia de performance é que para todas as listas L,

$$\mathrm{MMFD}(L) \leq (71/60 = 1.18333...)\mathrm{OPT}(L) + (31/6) \text{ bins}$$

e

$$R^{\infty}_{MMFD} = 71/60.$$

Em [45], há uma exposição longa da análise deste resultado de performance para MFFD.

De um modo geral, MFFD explora as situações no qual o empacotamento por FFD torna–se pobre; os dois algoritmos compartilham o mesmo bound de 71/60 quando os itens não excedem 1/2.

Descrição do Algoritmo:

Primeiramente a lista $L = \{a_1, a_2, ..., a_n\}$ é ordenada em ordem decrescente de tamanhos, $s(a_1) \geq s(a_2) \geq ... \geq s(a_n)$, como em FFD, e os seus itens classificados como:

$A = \{ x : s(x) \in (1/2,1]\}$	itens–A	tipo(x) = 1
$B = \{ x : s(x) \in (1/3,1/2]\}$	itens–B	tipo(x) = 2
$C = \{ x : s(x) \in (1/4,1/3]\}$	itens–C	tipo(x) = 3
$D = \{ x : s(x) \in (1/5,1/4]\}$	itens–D	tipo(x) = 4
$E = \{ x : s(x) \in (1/6,1/5]\}$	itens–E	tipo(x) = 5
$F = \{ x : s(x) \in (11/71,1/6]\}$	itens–F	tipo(x) = 6
$G = \{ x : s(x) \in (0,11/71]\}$	itens–G	

Em seguida, o algoritmo atua dentro de 5 fases distintas:

(1) Designa os itens–A aos primeiros $|A|$ bins em sequencia, tal que os níveis dos bins forme uma sequencia não–crescente. Esses bins são chamados de bins–A;

(2) Procede através dos bins–A da esquerda para a direita (i.e., do bin X_1 até o bin $X_{|A|}$), tratando o corrente bin X_i como segue: Se qualquer item–B não empacotado satisfizer em X_i, coloque dentro dele o maior de tais item–B que satisfaça. (Neste caso, só pode existir lugar para no máximo um deles)

(3) Proceda através dos bins–A da direita para a esquerda (i.e., do bin $X_{|A|}$ até o bin X_1), tratando o corrente bin X_i como segue:

Se X_i contém um item–B, nada faça.

Se os dois menores itens não empacotados de $C \sqcup D \sqcup E$ não satisfizer conjuntamente em X_i (ou se existe somente um de tais itens), nada faça.

Caso contrário, coloque o menor item não empacotado de $C \sqcup D \sqcup E$ em X_i, conjuntamente com o maior item não empacotado restante de $C \sqcup D \sqcup E$ que satisfaça;

(4) Proceda através dos bins–A uma última vez da esquerda para a direita, tratando o corrente bin X_i como segue: Se *qualquer* item não empacotado satisfizer em X_i, coloque dentro dele o maior de tais itens que satisfaça, repetindo este passo até que nenhum item não empacotado não satisfaça.

(5) Use *FFD* para empacotar os itens restantes nos bins iniciando com $X_{|A|+1}$.

As Figuras III.16 e 17, mostram situações nos quais um e outro algoritmo são superados e vice–versa.

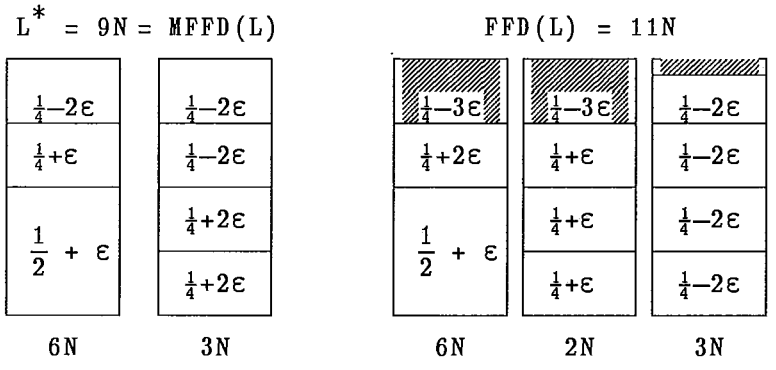


FIG. III.16: *Um empacotamento para o qual FFD(L)/MFFD(L) = 11/9.*

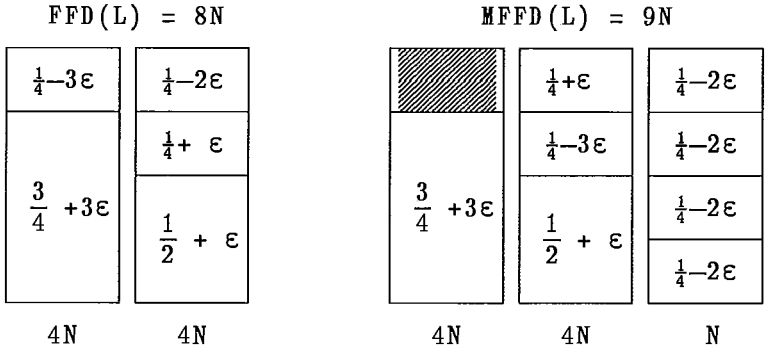


FIG. III.17: *Um empacotamento para o qual MFFD(L)/FFD(L) = 9/8.*

Porém, ainda nos dias de hoje, o algoritmo FFD é o mais utilizado em testes comparativos de algoritmos, porque já existe um considerável estudo sobre análise probabilística consolidando a eficiência desse algoritmo e mesmo, observe, na ausência de bins do tipo A os dois algoritmos são equivalentes. Além do que a complexidade de programação de MFFD é maior.

3.5 Variantes do Problema Clássico

Na literatura existem dois tipos de variantes, uma na qual a regra básica de empacotamento não segue o padrão inicial e a outra seguindo a mesma estrutura clássica, porém mudando a função objetivo do modelo. Várias heurísticas foram desenvolvidas especificamente para cada tipo de variante, juntamente com o estudo de performance do pior–caso da maioria delas. Existindo, portanto, uma grande quantidade de material na literatura. Nosso intuito, neste capítulo, será apresentar uma coletânea de muitas destas variantes para o caso uni–dimensional, procurando ser o mais abrangente possível. Muitos dos resultados de performance do pior–caso serão apenas citados.

3.5.1 Mudança nas regras básicas

Nesta seção abordaremos os resultados obtidos para as variantes do problema clássico uni–dimensional no qual o objetivo é ainda minimizar o número de bins usado, porém as regras básicas para o empacotamento são alteradas. Serão considerados três modificações do problema original:

(1) [KRAUSE, SHEN and SCHWETMAN, 50] Empacotamentos no qual limites são colocados a priori no número de ítens que podem ser empacotado num bin;

(2) [MAGAZINE and WEE, 57] Empacotamentos no qual uma ordem parcial é associada com o conjunto de ítens a serem empacotados e restritos a maneira no qual os ítens podem ser empacotados;

(3) [COFFMAN, GAREY and JOHNSON, 13] Empacotamentos no qual ítens podem chegar e deixar um bin dinamicamente.

A primeira modificação é tomada sobre um modelo para scheduling de multiprocessadores sob uma <u>única</u> restrição de recurso quando o número k de

processadores é fixado. Neste caso os ítens representam tarefas a serem executadas, com o tamanho de um ítem sendo a quantidade de recurso que ele consome de uma quantidade total C. Admitindo que todas as tarefas tenham todas a mesma unidade de comprimento de tempo de execução, então a schedule corresponde a designação de tarefas para tempos de início integral (bem definidos), tal que em nenhum instante existam mais do que k tarefas sendo executadas ou mais do que C quantidades do recurso sendo utilizadas. O objetivo é minimizar o tempo de início mais tarde. Isto corresponde ao **bin–packing** onde os tempos de início representam bins, cada um dos quais podendo conter no máximo k ítens.

Três algoritmos foram analisados para este problema, dois dos quais já conhecidos FIRST FIT e FIRST FIT DECREASING cujos limites no pior caso foram estabelecidos para o número de ítens por bin. Os resultados são:

$$\frac{27}{10} - \left\lceil \frac{37}{10k} \right\rceil \leq R_{FF}^{\infty} \leq \frac{27}{10} - \frac{24}{10k}; \quad R_{FFD}^{\infty} = 2 - \frac{2}{k}.$$

Quando $k \to \infty$, os limites ficam piores do que quando o número de ítens por bin não é restrito (*2.7* versus *1.7* e *2* versus *11/9*).

O outro algoritmo, de autoria dos pesquisadores logo acima mencionados, é o ITERATED LOWEST FIT DECREASING (*ILFD*), o qual primeiro coloca os ítens em ordem não–crescente por tamanho, como em *FFD*, e então inicia escolhendo algum limite inferior óbvio, digamos q, sobre *OPT(L)*. E supondo que tem–se q bins vazios, B_1, B_2,..., B_q, coloca a_1 em B_1 e procede através da lista de ítens, empacotando a_i num bin cujo conteúdo presente tem tamanho total mínimo (empate sendo resolvido pelo índice do bin, quando necessário). Uma iteração é terminada se sempre se atinge um ponto onde a_i não satisfaz em qualquer dos q bins (ou porque a capacidade C foi ultrapassada ou porque o limite k foi excedido). Neste caso, q é aumentado de *1* e nova iteração é tomada. Eventualmente o empacotamento completo será gerado para algum valor de q, e isto é a saída.

Foi mostrado que este algoritmo é mais lento embora atinja assintoticamente o limite de $4/3 = 1.333...$ (próximo do de *FFD*) quando não existe restrição no número de ítens por bin. O tempo de execução é de $O\ (nlog^2n)$ e o limite de performance provado foi $R^{\infty}_{I\ LFD} \leq 2$, sem determinação exata mas existindo conjectura de proximidade para $4/3$.

O segundo tipo de modificação adiciona uma ordem parcial ao conjunto L de ítens. É tomado sobre um modelo de "balanceamento de linha de montagem". Os ítens representam tarefas a serem executadas num único produto que se moverá ao longo de uma linha de montagem. Cada tarefa só pode ser excecutada em uma de uma seqüência de estações de trabalho B_1, B_2, ...etc (bins), e os tamanhos dos ítens corresponde aos tempos requeridos para execução das tarefas. Cada estação de trabalho dispõe de um período de tempo C para processamento de um produto, não podendo ser ultrapassado. Portanto um conjunto de tarefas pode ser designado para uma estação de trabalho (bin) se a soma de seus tempos requeridos (soma dos tamanhos dos ítens) não exceda C. O objetivo é minimizar o número de estações de trabalho. Obviamente, numa linha de montagem existe uma ordem de prioridades de tarefas. Neste caso, uma ordem parcial $\prec$ é interpretada como segue: $a_i \prec a_j$ significa que em qualquer designação de tarefas à estações de trabalhos (bins), a_i deve ser executado antes de a_j (com a possível inclusão que ambas as tarefas possam ser executadas na mesma estação de trabalho, desde que executando–se a_i antes de a_j dentro do tempo total C permitido); assim a_i pode ou ir num bin anterior ou no mesmo bin que a_j (veja Fig. III.16).

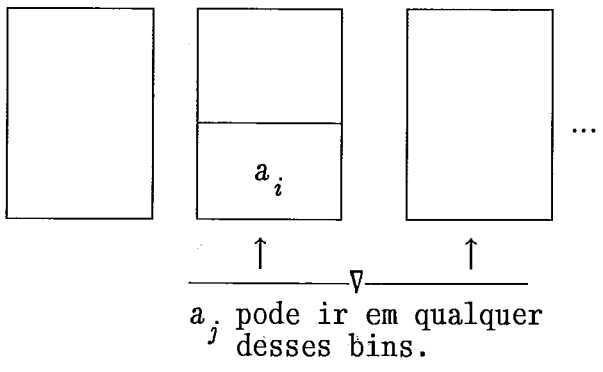


FIG. III.16 *Interpretação de* $a_i \prec a_j$.

Um algoritmo foi definido para este problema, o ORDERED FIRST FIT DECREASING (*OFFD*): Primeiramente, os ítens são ordenados pelo tempo que eles requerem em ordem não crescente, como em *FFD*. Então, preenche–se os bins em sequência. O bin B_i é empacotado colocando o maior ítem ainda não designado nele o qual a ordem parcial permitirá. O processo é repetido até que nenhum ítem mais possa ser colocado em B_i. A saída é quando todas as tarefas forem executadas para o único produto.

Foi mostrado, naquele trabalho, que $R^{\infty}_{OFFD} = 2$.

A terceira modificação considerada diz respeito ao problema o bin–packing dinâmico, o qual é uma extensão natural do modelo clássico uni–dimensional estático. Os ítens a ser empacotados são descritos por uma lista $L = \{a_1, a_2, \ldots, a_n\}$. Cada ítem (ou objeto) a_i em L corresponde a uma tripla (c_i, d_i, s_i), onde c_i é o tempo de chegada para a_i, d_i é o tempo de partida, e s_i é o seu tamanho. O ítem a_i reside no empacotamento por um intervalo de tempo $[c_i, d_i)$ e se admite que $d_i - c_i > 0$ para todo i. Um empacotamento é então uma designação de ítens à bins tal que em qualquer instante t e para qualquer bin B, os ítens designados para aquele bin os quais chegaram no instante t e ainda não o deixaram para àquele instante tem conteúdo total não maior do que a capacidade C. Admite–se, como antes, que cada s_i satisfaz $0 < s_i \leq C$.

Este modelo é motivado pela aplicação de alocação de memórias num computador, a qual é feita de maneira dinâmica. Os bins correspondendo a unidade de armazenagem, tais como páginas de memória do computador (ou discos fixos) e os ítens correspondendo a registros (arquivos) os quais devem ser armazenados dentro das páginas (discos fixos) por certos períodos de tempo especificados; sendo limitado no aspecto que registros não podem ser "sobrepostos" de uma unidade para a próxima. Este modelo pretende resolver o problema de como distribuir registros entre unidade de armazenamento tal que a todo tempo cada unidade tenha suficiente espaço para manter todo os registros designados para ela. Admite–se explicitamente que um registro pode sempre ser empacotado em qualquer unidade de armazenagem que tenha espaço total

disponível suficiente para ele, e portanto não considerando como os espaços são administrados dentro de uma unidade de armazenamento, evitando com isso que possa haver fragmentação.

O estudo foi restrito a análise de algoritmos cujas regras de empacotamento não movem ítens de um bin para um outro uma vez que eles tenham sido empacotados, e que operam *on–line*, i.e., os ítens são empacotados na medida que eles chegam sem conhecimento de chegadas futuras. O texto em si se concentra nos resultados obtidos para o FIRST FIT adaptado a esta situação. Através de argumentos tipo adversário, i.e., comparando os bounds com os de outros algoritmos on–line é mostrado que nenhum outro algoritmo on–line pode ter substancialmente melhor performance do que FIRST FIT. Os resultados foram os seguintes:

Para o caso em que o tamanho dos ítens é limitado por $0<s_i\leq C/2$, (obtidos através de meios analíticos)

$$1.75 \leq R^{\infty}_{FF}(1/2) \leq 1.78$$

e, para qualquer algoritmo on–line A

$$R^{\infty}_{A}(1/2) \geq 5/3 = 1.666...$$

No caso clássico, tínhamos

$$R^{\infty}_{FF}(1/2) = 1.5$$

o que mostra um enfraquecimento na performance deste algoritmo quando aplicado ao caso dinâmico.

Para o caso irrestrito, $0 < s_i \leq C$, a análise é mais complexa. Os limites encontrados foram menos apertados que antes, com

$$2.75 \leq R_{FF}^{\infty} \leq 2.897$$

e

$$R_{A}^{\infty} \geq 2.5$$

É interessante notar que no caso dinâmico o aumento do limite de performance para FF é de aproximadamente 55%, de cerca de *1.8* à *2.8*, o qual é um contraste com o aumento análogo de somente 13%, de *1.5* à *1.7*, para o caso do bin packing estático.

3.5.2 Mudança na Função Objetivo

Nesta seção abordaremos os resultados obtidos para as variantes do problema clássico uni–dimensional nas quais o objetivo não é mais minimizar o número de bins usados. Três variantes serão consideradas:

(1) [ASSMANN, JOHNSON, KLEITMAN and LEUNG, 1] Maximizando o número de bins acima de um dado nível.

(2) [FRIESEN and LANGSTON, 27] Minimizando os custos de empacotamento usando bins de tamanhos variáveis.

(3) [COFFMAN, LEUNG, and TING, 19] Maximizando o número de ítens empacotados.

A primeira variante considera o problema de empacotar os ítens de uma dada lista dentro de tantos bins quanto possível, sujeito a restrição que cada bin tenha um nível de enchimento de no mínimo um dado limiar $T>0$. Em outras palavras o objetivo é encher cada bin a um nível tão próximo quanto possível, mas não menor do que T. Este problema modela uma variedade de situações encontradas na indústria e comércio. Por exemplo, o enlatamento de produtos alimentícios satisfazendo um requisito mínimo de peso líquido rotulado na embalagem.

Esta variante é a versão dual do problema bin–packing. Vários algoritmos

foram apresentados proporcionando garantia de performance. Nós mencionaremos apenas um deles, o qual começa produzindo numa 1a. fase um empacotamento *FFD* de L para alguma dada capacidade $C > T$. Na 2a. fase do algoritmo os ítens são tomados iterativamente do último bin não–vazio e colocados no bin indexado mais baixo tendo um nível menor do que T; no fim deste estágio o empacotamento estará completo. A performance para esta regra, chamada $FFD(C)$, depende do valor de C escolhido para a fase 1. Foi mostrado em ASSMAMN et al. [1], usando como medida a razão invertida $OPT(L)/A(L)$, que

$$R^{\infty}_{FFD(C)} \geq \frac{3}{2} \quad para\ todo\ C \geq T$$

e

$$\lim_{C \to T} R^{\infty}_{FFD(C)} = 2$$

A segunda variante diz respeito ao problema no qual são dados uma coleção finita de tamanhos de bins com um suprimento ilimitado de bins de cada tamanho, e um custo de uso dos bins proporcional ao tamanho do bin. O objetivo sendo empacotar uma dada lista de ítens tal que o custo acumulado dos bins usados seja mínimo.

Foi analisado em [27] várias regras de aproximação; para a melhor delas foi provado um limite assintótico do pior–caso de $4/3$.

A última variante abordada trata "ainda" de um problema de maximização que fixa o número de bins e a sua capacidade. O objetivo, neste caso, é empacotar tantos ítens quantos forem possíveis dentro dos bins. Este problema tem uma ampla aplicação em pesquisa operacional, uma vêz que podemos imaginá–lo como um problema de "knapsacks múltiplos". Obviamente, trata–se de um problema NP–completo e daí o interesse em desenvolver e analisar algoritmos heurísticos rápidos para ele.

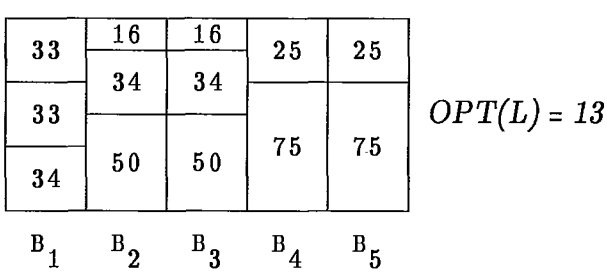
Formalmente o problema é o seguinte: Dado um conjunto de m iguais bins $B_1,\ldots,B_m$ e um conjunto de objetos organizados numa lista $L = (a_1,a_2,\ldots,a_n)$, como

empacotar dentro dos bins um subconjunto máximo de L tal que em nenhum bin a capacidade seja excedida.

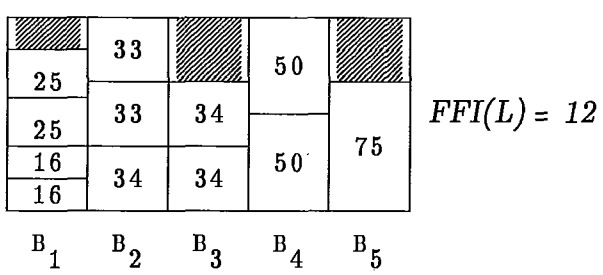
É intuitivo pensar, que qualquer algoritmo tendo razoável performance no pior–caso relativo a um algoritmo de otimização deva tentar empacotar um máximo subconjunto de pequenos objetos. Um algoritmo em particular, o FIRST FIT INCREASING (*FFI*), foi proposto e o limite de performance analisado. Este algoritmo faz o seguinte: primeiro ordena os itens em ordem não crecente por tamanho e então aplica a regra FIRST FIT até que um item na sequencia não satisfaça em qualquer dos bins; o que implica que nenhum dos itens restante não satisfará também. Portanto, com a regra *FFI* cada sucessivo objeto é colocado dentro do bin com menor índice no qual ele satisfaz. A Figura III.18 ilustra este procedimento. Para este foi mostrado que para qualquer lista L e qualquer número de bins m, tem–se $R^{\infty}_{FFI} = 4/3$.

Exemplo 4: Sejam $L = \{16,16,25,25,33,33,34,34,34,50,50,75,75\}$,

$C = 100$ e $m =$ número de bins $= 5$.



$OPT(L) = 13$



$FFI(L) = 12$

FIG. III.18: *Um exemplo de empacotamento FFI.*

CAPITULO IV

UMA CLASSE DE ALGORITMOS APROXIMATIVOS DECRESCENTES

4.1 A Classe proposta.

Uma quantidade grande de algoritmos aproximativos de bin–packing (Cf. [14]) têm sido, até agora, com raras exceções (e.g., algoritmo **Next Fit Decreasing**, **First Fit Increasing**), propostos e analisados considerando um preenchimento simultâneo dos bins, ou seja, os bins somente são considerados completos quando não existir mais nenhum elemento na lista para ser alocado. Isto provoca um efeito de dependência entre os bins. Entretanto, os bins são "livres" e cada um deles, em sequência, pode ser completado e descartado em qualquer etapa do empacotamento.

A classe que propomos corresponde a uma série de algoritmos aproximativos que assim procedem. O mais simples deles, o qual denominaremos de MMD "Maiores e Menores Decrescente", é um procedimento de alocação intuitivo e segundo os experimentos produz soluções tão boas quanto as do FFD (First Fit Decreasing), principalmente quando refinado, apresentando complexidade de tempo no pior–caso de $O(n)$ se ordenação é *pressuposta*.

A estratégia do algoritmo, em síntese, consiste em:

Passo 1. Ordenar a lista de objetos em ordem *decrescente* dos tamanhos de s_i, $i = 1,...,n$ e indexar os bins.

Passo 2. Empacotar os ítens, em sequência, do maior para o menor, num bin de índice mais baixo ainda não satisfeito, até onde puder ser feito nesta ordem; atualizando a lista após cada ítem empacotado. Se este preenchimento for parcial, ou seja, se a capacidade restante do bin (i. e., *a folga*) for maior do que o último elemento na lista, tomar outras decisões e/ou terminar de encher o bin com os ítens menores na sequência inversa da ordenação na lista, ou seja, do menor para o maior. Ir ao Passo (3).

Obs.: Tomando **outras decisões** antes de completar o bin com os "ítens menores", derivamos novos algoritmos, o qual denominaremos aqui de *refinados.*

Passo **3.** Se Lista = $\emptyset$, termine contabilizando o número de bins usados. Caso contrário volte ao Passo (2).

Deve ser notado, neste algoritmo que, na maior parte das vezes, a cada inspeção (i.e., comparação da folga resultante de um bin com o menor objeto da lista restante), os últimos objetos na lista cabem no espaço vazio do bin parcialmente usado, complementando–o e fazendo assim uma compensação ou balanço entre ítens maiores e menores. Isto torna o processo bastante rápido e interessante. Além disso, a cada iteração (Passo 2) um bin é finalizado, como pretendíamos, devido a pré–ordenação dos ítens na lista. Isto faz o algoritmo ser *on–line* com respeito aos bins. Proporcionando, portanto, uma vantagem operacional para *determinadas situações práticas* onde uma outra operação possa ser iniciada antes que o empacotamento total esteja concluido. **Neste caso, somente um bin fica ativo de cada vez, o que minimiza o custo total de utilização dos bins se existe um custo de permanência associado a cada bin aberto.**

A descrição do algoritmo proposto pode ser resumida como segue:

```
Procedure MMD;
< Definição das variáveis >
begin
        < Preordenação e Inicialização >  { ← Quicksort }
        Repeat
          Ative próximo bin  {nº de mochilas}
          While (Espaço Sufiente e Lista não vazia) do begin
                empacote objeto maior;
                atualize Lista
                end;
                < outras decisões >
          While (Espaço Suficiente e Lista não vazia) do begin
                empacote objeto menor;
                atualize Lista
                end;
        until (Lista vazia);
end;
```

Exemplo: Sejam $C = 60$ e $L = \{31(x2),17(x2),16(x3),13(x2),10\}$, então (veja Fig. IV.1), $MMD(L) = \frac{4}{3} \cdot OPT(L)$.

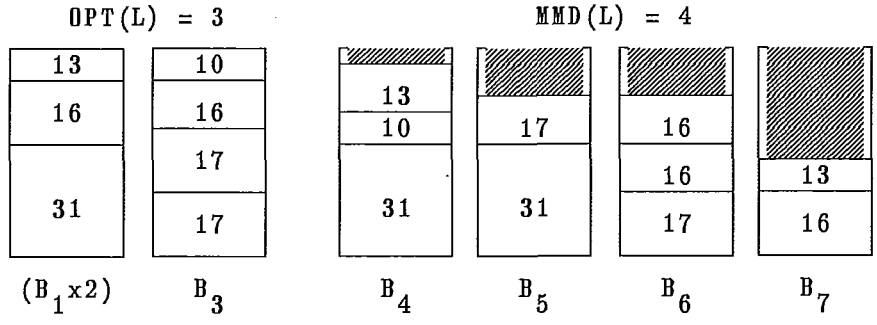


FIG. IV.1: *Um exemplo usando MMD sem refinamento.*

A seguir apresentamos três alternativas, para o Passo (2) do algoritmo, referente a **outras decisões**:

1a) Simplesmente termine de encher o bin, em questão, com os ítens menores na ordem inversa da ordenação na lista e vá ao Passo (3) do algoritmo proposto. (Neste caso, o algoritmo aproximativo é $O(n)$ se ordenação não é considerada, e para determinadas entradas poderá produzir resultados melhores que os da estratégia *FFD*. Como exemplos, observe Fig. IV.5 e Tabelas 10 e 11 na seção 4);

2a) Fazer uma rápida inspeção no restante da lista para encontrar algum ítem intermediário cujo tamanho complete a folga inteiramente. Se isto acontece, empacotá–lo, e ir ao Passo (3). Se não, proceder como antes, terminando o preenchimento do bin com os ítens menores.

3a) Fazer uma rápida inspeção para encontrar o ítem (ou ítens) intermediário mais próximo que caiba no bin, em questão. Logicamente, este ítem satisfaz o melhor possível (o processo funciona, neste instante, como um *Best–Fit* com respeito a lista de entrada). Se após o empacotamento deste ítem, a folga resultante é igual ao tamanho do último objeto da lista, empacote este último ítem também, encerrando o bin, e vá ao Passo (3) do algoritmo. Se a folga for maior, terminar de encher o bin com os ítens

menores e vá ao Passo (3); se for menor, o bin está encerrado e a etapa seguinte é ir, diretamente ao Passo (3) o qual termina o processo global ou segue para nova iteração. (Vê–se, naturalmente, que as decisões 2 e 3, são um **refinamento** do procedimento 1, o que produz resultados melhores, mas como conseqüência aumenta a complexidade computacional do algoritmo. Com a alternativa (3a) os resultados, raramente, diferem daqueles apresentados por FFD (veja Tabelas nos Gráficos 1 e 2).

Proposição 1. *A família dos algoritmos MMD é on–line com relação aos bins.*

Dem: Sejam P um empacotamento de $L = \{s_1, s_2, \ldots, s_n\}$, isto é, uma função

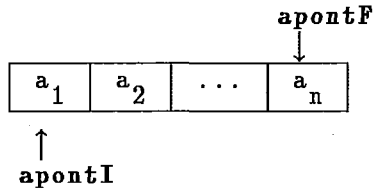
$$P: \{1,2,\ldots,n\} \rightarrow \{1,2,\ldots,n\},$$

tal que $\forall j,\ 1 \leq j \leq n,\ \Sigma_{P(i)=j} s_i \leq 1$ por MMD e, $\bar{s} \in L$ tal que $\bar{s}$ encerra o $BIN_{j'}$ $(1 \leq j' \leq j)$. Seja $L' \subseteq L$, a lista de objetos restantes após $\bar{s}$ ter sido designado. Então, por definição de MMD, isto significa que ou o empacotamento até $j' = j-1$ foi de tal maneira que a *folga* (i.e., o espaço disponível) deixada em cada BIN_j, é menor do que último item na lista L', ou $\exists x,\ x < \bar{s}$ em L que não tenha sido designado. Portanto, para algum $a_i \in L$ ainda não designado devemos ter $a_i \geq \bar{s} > folga(BIN_j,)$, $1 \leq j' < j$. Logo, considerando que os bins são dispostos da esquerda para à direita, o BIN_j que segue imediatamente ao BIN_j, é o escolhido, dado que ele está vazio e a_i, próximo item a ser designado, não cabe em qualquer bin à sua esquerda; caracterizando, portanto, o algoritmo como **on–line** em relação aos bins. ∎

4.1.2. Complexidade Computacional.

Afirmação 1: *Todas as instruções, excluindo a parte de ordenação e outras decisões, são executadas no máximo* ***n*** *vezes.*

Dem: Suponha que existam dois apontadores sobre a lista L, apontI e apontF, os apontadores inicial e final, respectivamente.



Desta forma obtemos a seguinte estrutura equivalente, bem simples, para o algoritmo anteriormente proposto:

Algoritmo Maiores_Menores_Decrescente
Entrada: $L = \{s_1,...,s_n\}$, onde $0 < s_i \leq 1$ para $1 \leq i \leq n$.
Saída: O número do bin dentro do qual s_i é colocado.

```
procedure MMD(S: RealArray; n: integer; var bin: IntegerArray);
var
        usado: array[1..n] of real;
        {usado[j] é o espaço já ocupado no bin j}
        i,j,apontI,apontF: Integer;
begin
        {S em ordem decrescente pressuposto}
        for j := 1 to n do usado[j] := 0;
        j := 0; apontI := 1; apontF := n;  {Inicialização}
        Repeat
                Inc(j);          {nº de mochilas}

                while usado[j] + s[apontI] ≤ 1 and apontI ≤ apontF do begin
                        empacote objeto s[apontI];
                        Inc(apontI)
                        end;

                while usado[j] + s[apontF] ≤ 1 and apontI ≤ apontF do begin
                        empacote objeto s[apontF];
                        Dec(apontF)
                        end

        until (apontI > apontF);
end
```

Observando–se o algoritmo agora, nota–se que as duas instruções **while** são complementares, significando que a cada iteração de j os dois (ou apenas um) apontadores **apontI** e **apontF** caminham em sentido opostos na lista L. Entretanto, quando eles se ultrapassam, fica determinado um valor $j = \bar{j}$, $1 \leq \bar{j} \leq n$, solução do problema e, cada instrução é executada no máximo $\lfloor n/k \rfloor$–vezes, para algum $k \in \{1,...,n\} \subset \mathbb{N}$. O caso extremo sendo atingido quando $k = 1$ ou $\bar{j} = 1$, i.e., apenas uma mochila é usada.

Por outro lado, se $\bar{j} = n$ (i.e., forem usadas n mochilas, cada uma com um único objeto) teremos o **repeat** sendo testado n vezes, ocasionando novamente o número máximo de utilização de uma instrução. ■

Afirmação 2: *As complexidades de pior e melhor casos para MMD são ambas O(n). Portanto, o algoritmo é o melhor possível, pois, no mínimo são necessárias n operações para empacotar n itens.*

Dem: Pela afirmação 1, cada instrução é executada no máximo n vezes. A mecânica do algoritmo se resume ao bloco:

```
Repeat
    Ative próximo bin;  {nº de mochilas}
        While (Espaço Suficiente e Lista ≠ ∅) do begin
            empacote objeto maior;
            atualize Lista
            end;
        While (Espaço Suficiente e Lista ≠ ∅) do begin
            empacote objeto menor;
            atualize Lista
            end;
until (Lista = ∅);
```

Os dois While referem–se ao preenchimento dos bins com os ítens maiores e menores, respectivamente. Então, qualquer que seja a instância de entrada o 2º while complementa o 1º while. Assim, se o 1º while for executado k–vezes, $k = 1,...,n$, o 2º executará k'–vezes, $k'=n-k$, fornecendo uma complexidade do algoritmo no pior–caso da ordem de $O(k+k') = O(n)$. Por outro lado, como deve ser notado, este algoritmo não é sensível à instâncias de entrada, sempre efetuando $O(n)$ comparações, mesmo no melhor caso. ■

Considerando em **outras decisões** uma *busca sequencial* para escolher elementos intermediários, e usando como estrutura de dados uma lista duplamente

encadeada, teremos o algoritmo exposto numa forma mais simples, porém funcionando em $O(n^2)$. Por outro lado, usando uma estrutura mais pesada, tipo árvore B–Tree, o tempo computacional no pior–caso, dos algoritmos refinados, cai para $O(nlogn)$, uma vez que a busca à cada elemento intermédiario é feita no pior das hipótese em $O(logn)$.

4.1.3 Performance do Algoritmo Proposto sem Refinamento.

Conforme mostrado no Cap. III, parágrafo 3.2, se L está em ordem decrescente e A é qualquer algoritmo ANY FIT (AF), então o comportamento do pior–caso assintótico para AD (i.e., o algoritmo decrescente) encontra–se dentro de um limite estreito dado por:

$$11/9 \leq R^{\infty}_{AD} \leq 5/4 \ .$$

Note que algoritmos–AFD sempre coloca os ítens, um de cada vez, nos bins tomando–os em ordem sequencial na lista, do maior para o menor.

Os algoritmos FIRST FIT (FF) e BEST FIT (BF), por exemplo, são algoritmos ALMOST ANY FIT [$\subseteq$ ANY FIT] e, como já é bem conhecido $R^{\infty}[FFD] = R^{\infty}[BFD] = 11/9$.

O algoritmo WORST FIT (WF) o qual coloca a_i num bin aberto com a maior folga (empate sendo decidido em favor do bin mais à esquerda), é ANY FIT mas não ALMOST ANY FIT. Como exemplo, sejam os seguintes dados:

$$L = \{a_1=9, a_2=2, a_3=8, a_4=2, a_5=7, a_6=1\} \text{ e } C = 14,$$

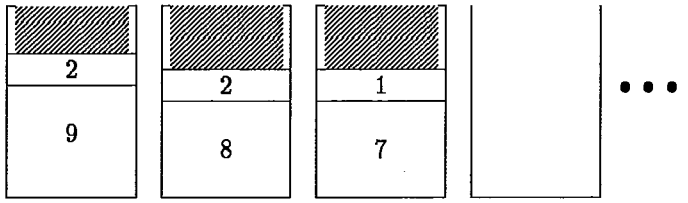


FIG. IV.2: *Exemplo com a Heurística Worst–Fit.*

Como se pode notar, o item a_4 satisfaz no BIN_1 mas não é colocado lá, o que quebra a regra *AAF*, porém continua sendo *AF* porque a_4 foi designado a um bin não–vazio qualquer com maior folga, dentro do qual ele satisfaz.

A performance do pior–caso de *WFD* não foi explicitamente determinada, sabendo–se apenas que se encontra dentro da faixa estabelecida anteriormente para os algoritmos *AD*.

O algoritmo Next Fit Decreasing (*NFD*) é um exemplo de um algoritmo que não pertence a classe dos algoritmos ANY FIT DECREASING (*AD*). Sua performance absoluta no pior–caso é $NFD(L) \approx 1.691 \cdot OPT(L) + 3$ *bins* (Cf. BAKER & COFFMAN JR. [2]).

Proposição 2. *O algoritmo MMD* **sem** *pré–ordenação, não pertence a classe dos algoritmos ANY FIT.*

Dem: Seja o seguinte contra–exemplo[2] :

$$L = \{a_1=9, a_2=2, a_3=14, a_4=2, a_5=7, a_6=1\} \text{ e } b = 15,$$

então $BIN_1 = \{a_1, a_2, a_6\}$, $BIN_2 = \{a_3\}$ e $BIN_3 = \{a_4, a_5\}$; o item a_4 cabe no BIN_1 mas é colocado no BIN_3 (naquele momento, o bin vazio mais à esquerda), o que quebra a regra básica ANY FIT. ■

[2]Aqui admitimos que o comprimento n da lista L é conhecido à priori e os bins estão dispostos da esquerda para a direita. Tais algoritmos são ditos *fechados* [12].

Note que a designação de "maiores e menores" anteriormente colocada ao nome do algoritmo perde o seu sentido aqui, e o mais correto agora seria denominar este último algoritmo de *ES* (Extremos que Satisfazem ou preenchem um bin).

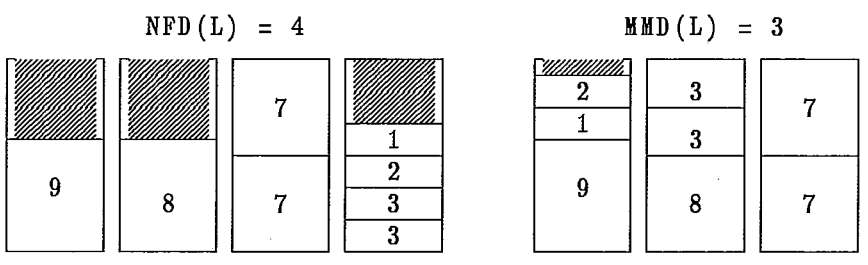
Proposição 3. *MMD(L) ≤ NFD(L) qualquer que seja L.*

Dem: Suponha que $MMD(L) > NFD(L)$. Assim, sejam B_j, $1 \le j \le k$, os bins empacotados pela regra *NFD* de uma lista L ordenada, e B'_j, $1 \le j \le k+l$ $(l = 1,..., n-k)$, os bins empacotados da mesma lista por *MMD*. Sem perda de generalidade, seja α um elemento de B'_{k+1}. Por definição de *MMD*, $\alpha > gap(B'_j)$, $1 \le j \le k$. Isto é, α não satisfaz em qualquer bin descartado, anterior a B_{k+1}. Por outro lado, por construção de *MMD* (veja Fig. IV.3 (i) e (ii)), em cada iteração,

$$cont(B'_j) = \begin{cases} cont(B_j) \\ \quad \text{ou} \\ cont(B_j) + \text{"itens menores"} \\ \quad \text{ou} \\ cont(B_j) - \text{"itens menores" deslocados para bins anteriores;} \end{cases} \qquad (1 < j \le k)$$

Como *NFD* empacota a mesma lista L, $\alpha \in cont(B_j)$ para algum $1 \le j \le k$. Assim devemos ter também que $\alpha \in cont(B'_j)$, para $1 \le j \le k$. Uma contradição ! ■

(i) L = {9,8,7,7,3,3,2,1} e C = 14.



(ii) L = {8,7,6,5,4,3,2,1} e C = 14.

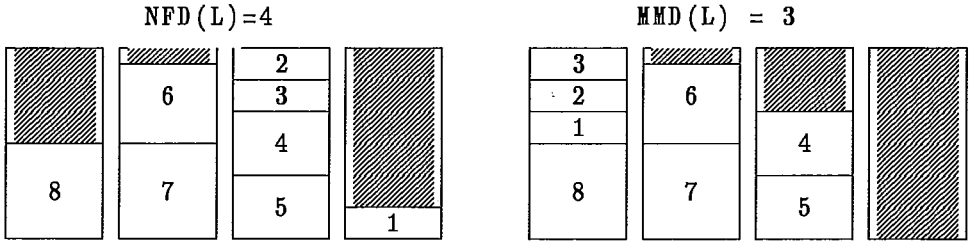


FIG. IV.3 (i) e (ii): *Exemplos comparativos entre NFD e MMD.*

Para mostrar que os outros algoritmos da classe MMD também satisfazem a relação da proposição 3, usamos o seguinte raciocínio: **Para provar um resultado (bound) sobre um algoritmo mais sofisticado *A*, primeiro mostramos que o ressultado desejado vale para um algoritmo menos sofisticado *B* (e mais fácil de analisar), e então mostramos que $A(L) \leq B(L)$ para todas listas *L*.**

Designaremos por *MMDr–k* o algoritmo proposto com k refinamentos (i.e., k elementos intermediários).

Proposição 4. *$MMDr(L) \leq NFD(L)$ para todas listas L.*

Dem: Seja $MMDr^*$ a versão restrita de *MMDr*. Ela difere de *MMDr* no fato que, tão logo um bin receba um item intermediário ele é declarado *encerrado* e não pode receber mais nenhum item. Obviamente, $MMDr^*(L) \leq NFD(L)$ e, consequentemente, $MMDr(L) \leq NFD(L)$ para qualquer que seja L. ■

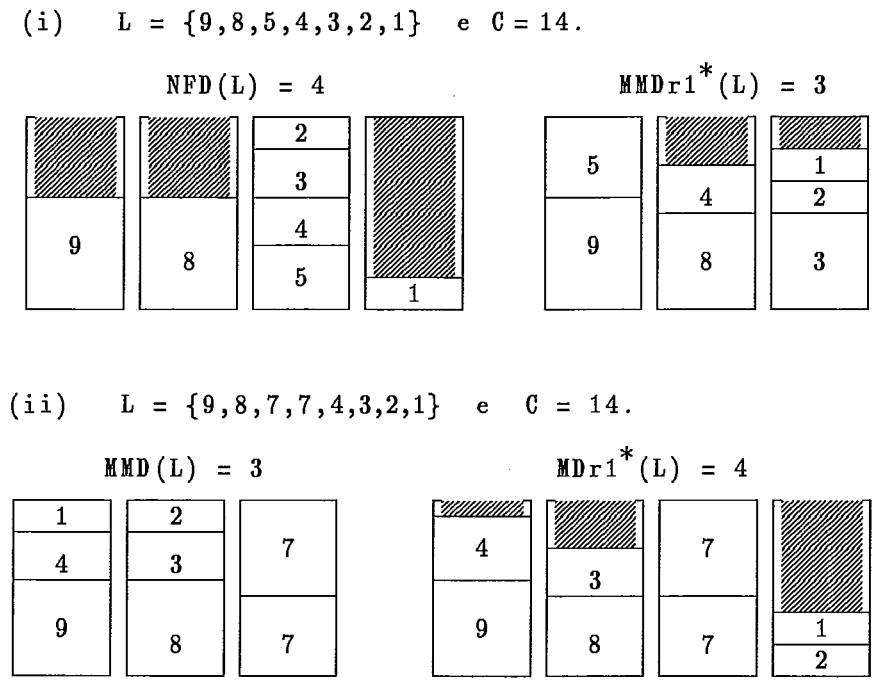


FIG. IV.4: *Exemplos no qual NFD(L)≥MMDr1*(L)* e *MMDr1*(L)≥MMDr1(L).*

CONJECTURA. *Para a classe dos algoritmos do tipo MMD,*

$$11/9 \leq R^{\infty}_{C_{MMD}} \leq 1.691$$

A Fig. IV.5 mostra um exemplo de lista assintótica no qual o limite inferior para o pior–caso da classe *MMD* é quase atingido. Além disso, mostra também que nem sempre *FFD* é melhor do que *MMD*.

L = { N repetições de $[(\frac{1}{2}+\epsilon)\times 6, (\frac{1}{4}+\epsilon)\times 6, (\frac{1}{4}-2\epsilon)\times 12]$ },
onde $0<\epsilon<\frac{12}{2}$ e $n = 30N$.

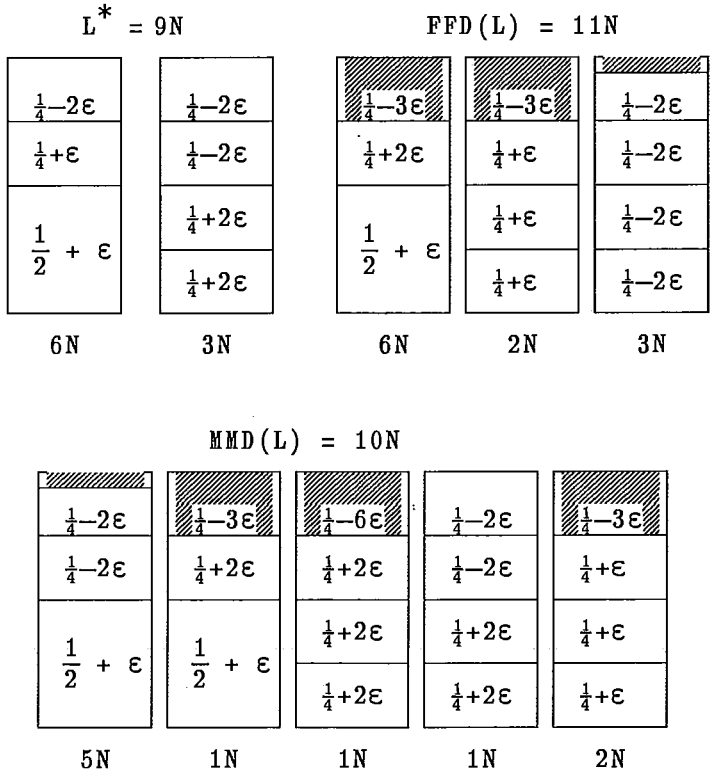


FIG. IV.5: *Um empacotamento para o qual FFD(L)/L* = 11/9 e MMD(L)/L* = 10/9.*

O limite superior na conjectura é válido conforme ficou provado nas proposições 3 e 4, e ilustrado na Fig. IV.4 (i) e (ii). ■

4.1.4 Uma Melhoria em Maiores e Menores Decrescentes (*MMDat*)

Se nós considerarmos que os dados gerados para a Lista $L \to \infty$, sejam de tal sorte que os seus valores são independentes e identicamente distribuidos (i.i.d), produzindo uma distribuição uniforme U(0,1], após ordenação, teremos um gráfico semelhante ao da Fig. IV.6. (A capacidade C dos bins é, essencialmente, um fator de escala, assim fazemos a normalização usual $C = 1$).

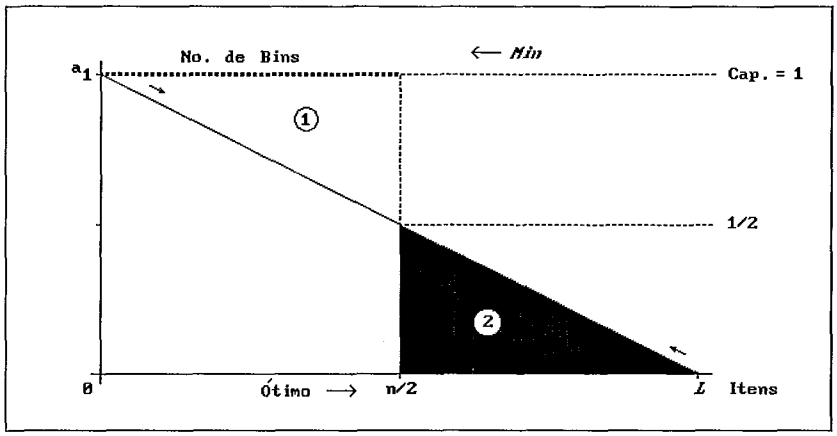


FIG. IV.6: *Aplicação de MMD para* $a_1 = 1$.

Quando *MMD* é executado, os pontos sobre a área sombreada (2) são transferidos para a região (1), produzindo um empacotamento perfeito de $\frac{n}{2}$ bins ($n \to \infty$, um inteiro par). Por outro lado, a função que determina a reta de distribuição dos tamanhos dos itens é dada por

$$f(x) = -\left[\frac{a_1}{n}\right]x + a_1$$

onde $f(n) = 0$ e $f(0) = a_1$. Logo,

$$x^* \approx \lim_{n \to \infty} \sum_{i=0}^{n} a_i = \int_0^n f(x)dx = \int_0^n (-\frac{a_1}{n})xdx + \int_0^n a_1 dx = \frac{n}{2},$$

é o número ótimo de bins. Assim, quando $a_1 = 1$ e $n \to \infty$, teremos

$$\lim_{L \to \infty} \frac{MMD(L)}{x^*} \to 1$$

A aplicação direta do algoritmo *MMD* para $a_1 < 1$, acarreta como resultado um maior número de bins, pois os pontos sobre a área hachuriada que são transferidos, não têm um encaixe perfeito na região (1), conforme mostra a Fig. IV.7. É necessário, então, uma correção para contornar isto.

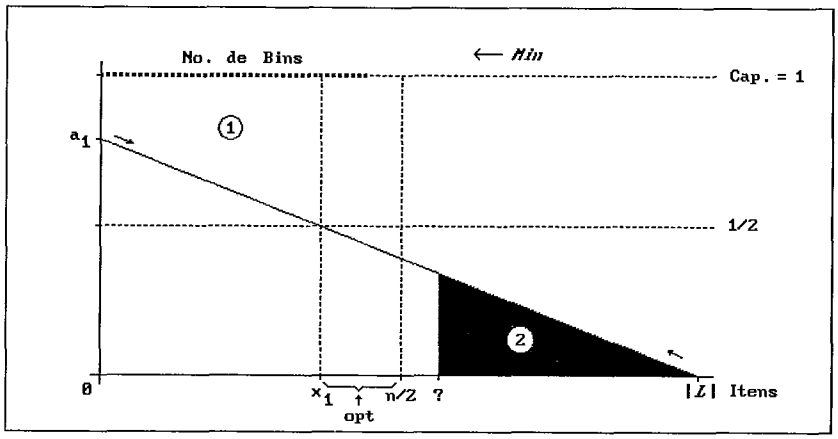


FIG. IV.7: *Aplicação de MMD para* $a_1 < 1$.

A Fig. IV.8 mostra uma melhoria de *MMD* pela utilização dos limites x_1 e x_2. As setas indicam o sentido que o algoritmo atua.

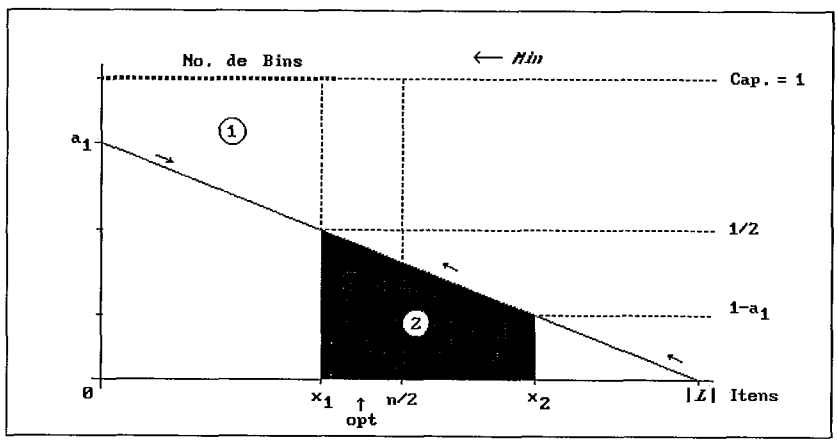


FIG. IV.8: *Aplicação de MMDat para* $a_1 \leq 1$.

Tecnicamente, a estrutura do algoritmo passa a ser:

Rep e at
    While 1 {como antes em *MMD*}
    While 2 {atua entre x_1 e x_2, ordem crescente sequencial}
    While 3 {como antes}
unt i l;

A complexidade deste algoritmo permanece ainda sendo *O(n)*, desde que cada elemento necessita ser examinado somente uma vez.

Os valores de x_1 e x_2 são, facilmente, obtidos por:

$$x_1 = n\left[1 - \frac{1}{2a_1}\right] \quad \text{e} \quad x_2 = n\left[2 - \frac{1}{a_1}\right]$$

Note ainda que, quando $a_1 = 1$,

$$x_1 = \frac{n}{2} \quad \text{e} \quad x_2 = n.$$

Este algoritmo, portanto, é uma generalização do anterior. Quando $a_1 = \frac{1}{2}$, temos $x_1 = x_2 = 0$. A partir daí, i.e, para $a_1 < 1/2$ não usamos mais x_1 e x_2 como limites.
(Cf. resultados na Tabela 4 e Gráfico 5, para uma constatação de melhoria na prática).

A Figura IV.9 mostra um exemplo típico de dados gerados aleatoriamente, através do computador.

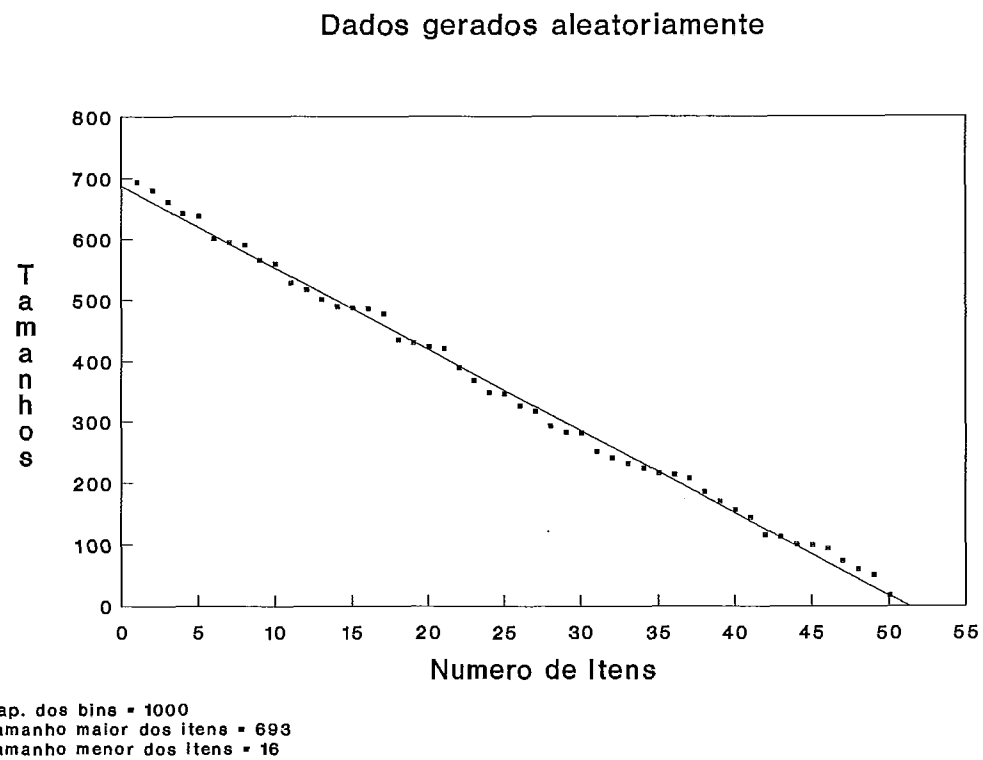


FIG. IV.9

4.1.5. Algoritmos Refinados.

MMDR–k é o algoritmo proposto com k refinamentos (i.e., k elementos intermediários que melhor satisfazem). Obviamente, se k for de tal sorte que não seja necessário percorrer a lista na ordem inversa, teremos o algoritmo análogo ao *FFD* para esta colocação do problema, e portanto, com uma razão de performance assintótica de *11/9*. A complexidade de tempo no pior–caso, no entanto, será de *O(nlogn)* com ou sem pré–processamento, usando estrutura de dados tipo árvores B–Tree. Tendo, contudo, o cuidado de testar, para acelerar o processo em média (principalmente para listas de entrada de dados muito grandes), se para cada elemento intermediário introduzido no bin ativo, a folga resultante desaparece quando adicionalmente a este anexamos o último item da lista (i.e, previamente fazemos uma inspeção), obteremos um algoritmo o qual denominaremos de *Progressivo Decrescente* (*PD*) que produz empacotamento semelhante ao algoritmo *First–Fit Decreasing* (*FFD*), a menos de alguns elementos repetidos colocados em bins diferentes. Por exemplo, seja L a seguinte lista ordenada: s_1=12; s_2=10; s_3=7; s_4=5; s_5=3; s_6=s_7=s_8=2. Seja a capacidade dos bins igual a 14, então os seguintes empacotamentos seriam produzidos:

FFD : s_1 e s_6 no BIN_1; s_2,s_5 no BIN_2; s_3,s_4,s_7 no BIN_3 e s_8 no BIN_4

PD : s_1 e s_6 no BIN_1; s_2,s_5 no BIN_2; s_3,s_4,s_8 no BIN_3 e s_7 no BIN_4

Ressaltamos, que o **mecanismo** deste algoritmo heurístico para preenchimento de um *bin* é um método guloso e **lembra** o do clássico algoritmo de J. B. KRUSKAL (1956) para a determinação da *Árvore Geradora Mínima* (MST). A lembrança com aquele algoritmo, que pode ser chamado o *smallest–edge–first*, é como segue: Primeiro ordenamos todas as linhas (itens) na rede dada por peso (tamanho dos itens), em ordem crescente (decrescente). Então uma a uma as linhas (itens) são examinadas em ordem, do menor para o maior (do maior para o menor). Se uma linha i (item), sob examinação, é achada para formar um ciclo (ultrapassar a capacidade do bin), quando adicionada as linhas (itens) já selecionadas, ela é ignorada (deixado para os próximos bins). Caso

contrário, a linha i é selecionada para ser incluída na MST (bin ativo). Como na Árvore Geradora Mínima, nem todos os ítens precisam ser examinados porque *a folga* do bin, em cada instante, é sempre comparada com o último ítem da lista, e quando menor, os ítens restantes da lista não precisam mais ser examinados para aquele bin ativo. Este processo é repetido, para as linhas restantes na árvore, i.e., uma nova árvore geradora mínima (bin ativo) é procurada (satisfeito) para o **grafo restante**, até que todas as linhas (ítens) tenham sido designadas (empacotados).

Obviamente, se fosse possível estabelecermos uma analogia entre estes dois problemas e tivéssemos uma transformação polinomial entre eles, estaríamos provando que $P = NP$. Tendo em vista que o algoritmo de Kruskal determina uma MST em tempo polinomial.

O que acontece porém, é que o processo de formação da MST, isto é, o mecanismo de construção da MST pode ser transportado, mas não há ligação em que determinar a árvore geradora mínima ou detectar um ciclo signifique satisfazer um bin. O ciclo no grafo apenas identifica um bin em fase de preenchimento e a MST pode muito bem significar que a capacidade do bin foi ultrapassada, mesmo sem formar ciclo.

4.1.6. Resultados Computacionais.

Uma bateria de testes computacionais indicam que as soluções encontradas por estas novas heurísticas são, para muitos problemas, bem próximas do ótimo. Isto nos dá, portanto, suporte para viabilizar os métodos.

Os algoritmos foram implementados em Pascal, num micro–computador IBM PC/AT(386), e testado em dados gerados aleatoriamente e dados produzidos a partir de empacotamentos ótimos conhecidos. Este último recurso é mostrado através da procedure GERADOR mais adiante.

A seguir apresentamos os resultados computacionais dos algoritmos, através de gráficos ilustrativos de performance e tabelas, correspondentes. A notação empregada nos algoritmos é:

FFD — First Fit Decreasing;

MMD — Maiores e Menores Decrescente;

MMDat — Maiores e Menores Decrescente atualizado;

MMDr1 — MMD com 1 ítem intermediário que melhor satisfaz o bin;

MMDif — MMD com 1 ítem intermediário que satisfaz o bin plenamente;

MMDmf — MMD com 1 ítem intermediário que forçosamente deixa uma folga no bin para ser preenchida pelos ítens menores;

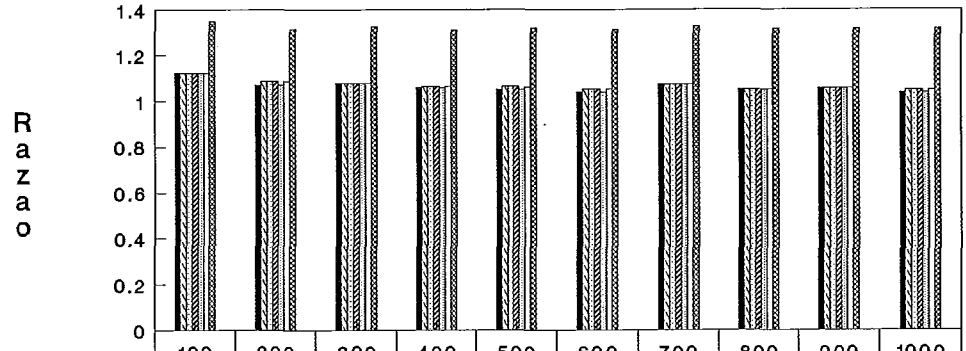
NFD — Next Fit Decreasing.

Obs: Todos estes algoritmos da classe *MMD* poderiam ser melhorados mais ainda se no retorno com os ítens menores, usássemos uma tabela hashing para verificar se existe algum ítem intermediário que preencha exatamente o bin. Se não, continuaríamos empacotando via ítens menores, sendo que a cada ítem menor empacotado, novamente a tabela hashing seria consultada, e assim por diante até terminar de encher um bin. O processo é repetido para os demais bins. Evitamos colocar os resultados referentes a esta afirmação porque julgamos que o estudo ficaria por demais longo e cansativo, além do que a conclusão não seria muito diferente.

Os Gráficos 1 e 2, a seguir, mostram as performances do pior–caso e média dos algoritmos para 50 testes realizados em cada uma das instâncias geradas, randomicamente, no intervalo (0,1]. A capacidade dos bins foram todas feitas iguais a 1.

GRAFICO 1

Razao de Performance Maxima

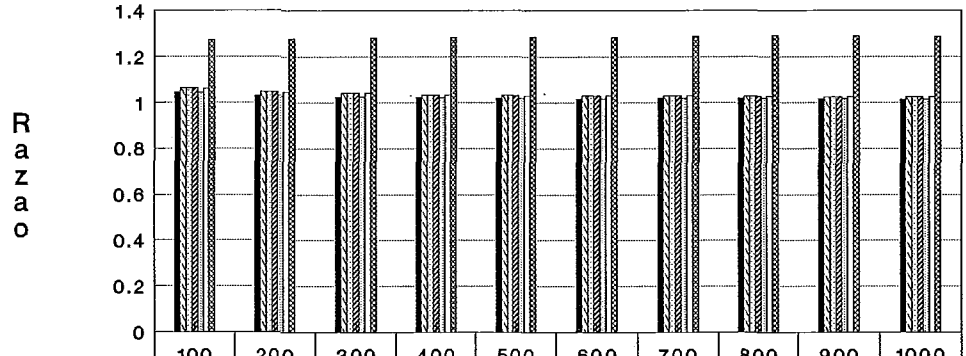


	100	200	300	400	500	600	700	800	900	1000
FFD	1.121	1.069	1.077	1.057	1.051	1.039	1.073	1.05	1.054	1.035
MMD	1.121	1.087	1.077	1.063	1.064	1.053	1.073	1.05	1.054	1.047
MMDat	1.121	1.087	1.077	1.063	1.064	1.053	1.073	1.05	1.054	1.047
MMDif	1.121	1.087	1.077	1.063	1.064	1.053	1.073	1.05	1.054	1.047
MMDr1	1.121	1.069	1.077	1.057	1.051	1.039	1.073	1.05	1.054	1.035
MMDmf	1.121	1.087	1.077	1.063	1.06	1.053	1.073	1.05	1.054	1.047
NFD	1.348	1.313	1.326	1.309	1.32	1.311	1.324	1.315	1.314	1.314

Ordem da Instancia

GRAFICO 2

Razao de Performance Media



	100	200	300	400	500	600	700	800	900	1000
FFD	1.043	1.029	1.026	1.022	1.019	1.017	1.019	1.018	1.016	1.013
MMD	1.063	1.047	1.04	1.034	1.032	1.029	1.029	1.028	1.025	1.025
MMDat	1.063	1.047	1.04	1.034	1.032	1.029	1.029	1.028	1.025	1.025
MMDif	1.063	1.047	1.04	1.034	1.031	1.029	1.029	1.028	1.025	1.025
MMDr1	1.043	1.029	1.026	1.022	1.019	1.017	1.019	1.018	1.016	1.013
MMDmf	1.062	1.047	1.04	1.034	1.031	1.029	1.029	1.028	1.025	1.025
NFD	1.271	1.279	1.282	1.284	1.285	1.284	1.289	1.291	1.288	1.286

Ordem da Instancia

Nas Tabelas 3,4,5 e 6, consideramos instâncias onde a magnitude dos ítens foram uniformemente distribuídas no intervalo (0,1]. Rodamos 50 testes para cada tamanho do problema (i.e, n e C fixados). Como o valor ótimo de cada instância não estava disponível, medimos a performance de cada algoritmo calculando a razão

$$\frac{\textit{solução heurística}}{\textit{limite inferior para solução ótima}}$$

onde o *limite inferior* ou *ótimo de referência* foi obtido por $\lceil \Sigma\, a_i/C \rceil$. Esta razão é, obviamente, um limite superior na performance do algoritmo.

Os Gráficos 3,4,5,6 e 7, ilustram o desempenho de apenas 3 algoritmos, FFD, MMD e NFD, por causa do nosso interesse especial em justificar o bom desempenho do algoritmo MMD. O desempenho mostrado, refere–se as razões máxima e média de 50 testes realizados. A fim de não haver indagações de que, poderia ter havido uma coincidência entre FFD e MMD, expomos o Gráfico 4 com razão média dos 50 testes realizados. Há, como se pode notar, uma uniformidade nos resultados obtidos. No Gráfico 5, fixamos o número de ítens em 100 (pois foi o que apresentou o pior desempenho médio para MMD em relação a FFD) e gradativamente aumentamos o número de testes para 300. O objetivo é mostrar que o número de testes é irrelevante, uma vez que a relação de proximidade ou afastamento de um algoritmo em relação ao outro permanece quase constante.

De outro lado, no Gráfico 6, mostramos que à medida que a lista de ítens aumenta, conservando–se o número de testes fixado em 50, o afastamento dos resultados dos algoritmos, MMD e FFD, fica muito próximo, enquanto os de Next Fit Decrescente (NFD) cresce quase linearmente. Isto mostra o bom desempenho do algoritmo MMD.

O Gráfico 7, revela o ajuste do algoritmo MMD quando a capacidade dos bins é inferior a 1. O algoritmo permanece $O(n)$.

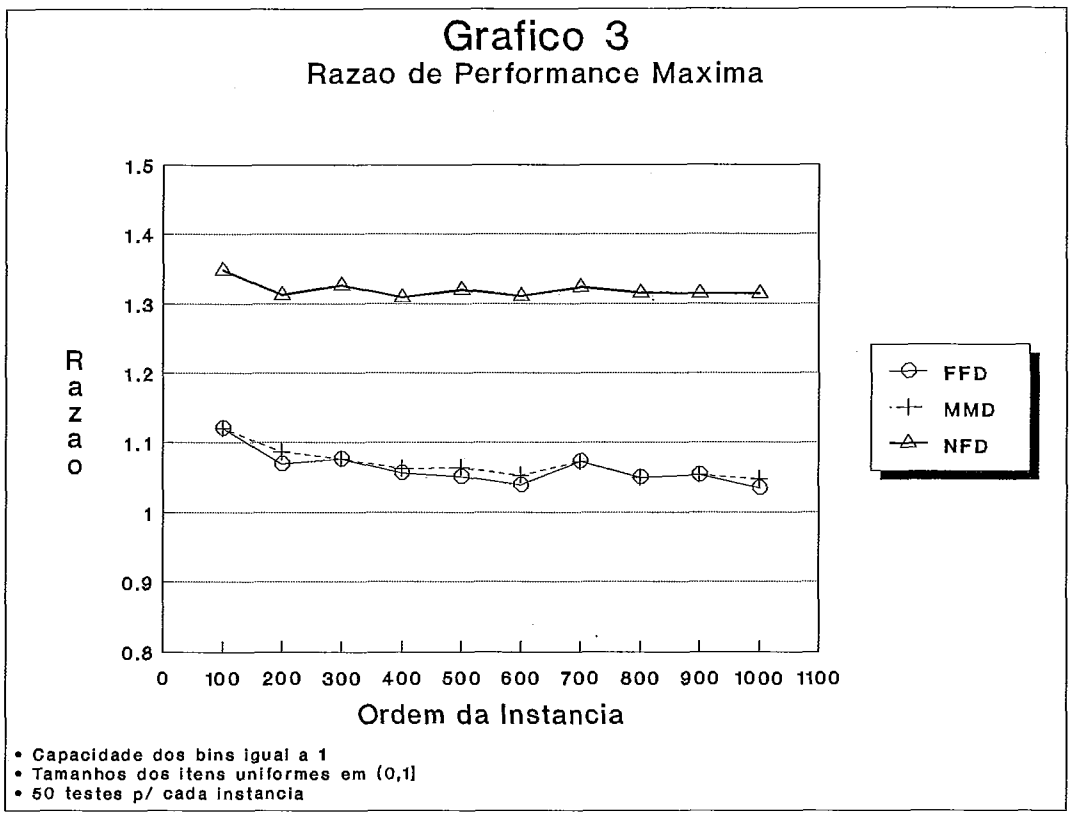


TABELA 3

Problemas com tamanhos uniformes em (0,1]
Performance Maxima (Frequencia do otimo) : (Cap. dos Bins = 1)

it	n	FFD	MMD	MMDat	MMDif	MMDr1	MMDmf	NFD
50	100	1.121(3)	1.121(0)	1.121(0)	1.121(0)	1.121(3)	1.121(0)	1.348(0)
	200	1.069(0)	1.087(0)	1.087(0)	1.087(0)	1.069(0)	1.087(0)	1.313(0)
	300	1.077(0)	1.077(0)	1.077(0)	1.077(0)	1.077(0)	1.077(0)	1.326(0)
	400	1.057(0)	1.063(0)	1.063(0)	1.063(0)	1.057(0)	1.063(0)	1.309(0)
	500	1.051(0)	1.064(0)	1.064(0)	1.064(0)	1.051(0)	1.060(0)	1.320(0)
	600	1.039(0)	1.053(0)	1.053(0)	1.053(0)	1.039(0)	1.053(0)	1.311(0)
	700	1.073(0)	1.073(0)	1.073(0)	1.073(0)	1.073(0)	1.073(0)	1.324(0)
	800	1.050(0)	1.050(0)	1.050(0)	1.050(0)	1.050(0)	1.050(0)	1.315(0)
	900	1.054(0)	1.054(0)	1.054(0)	1.054(0)	1.054(0)	1.054(0)	1.314(0)
	1000	1.035(0)	1.047(0)	1.047(0)	1.047(0)	1.035(0)	1.047(0)	1.314(0)

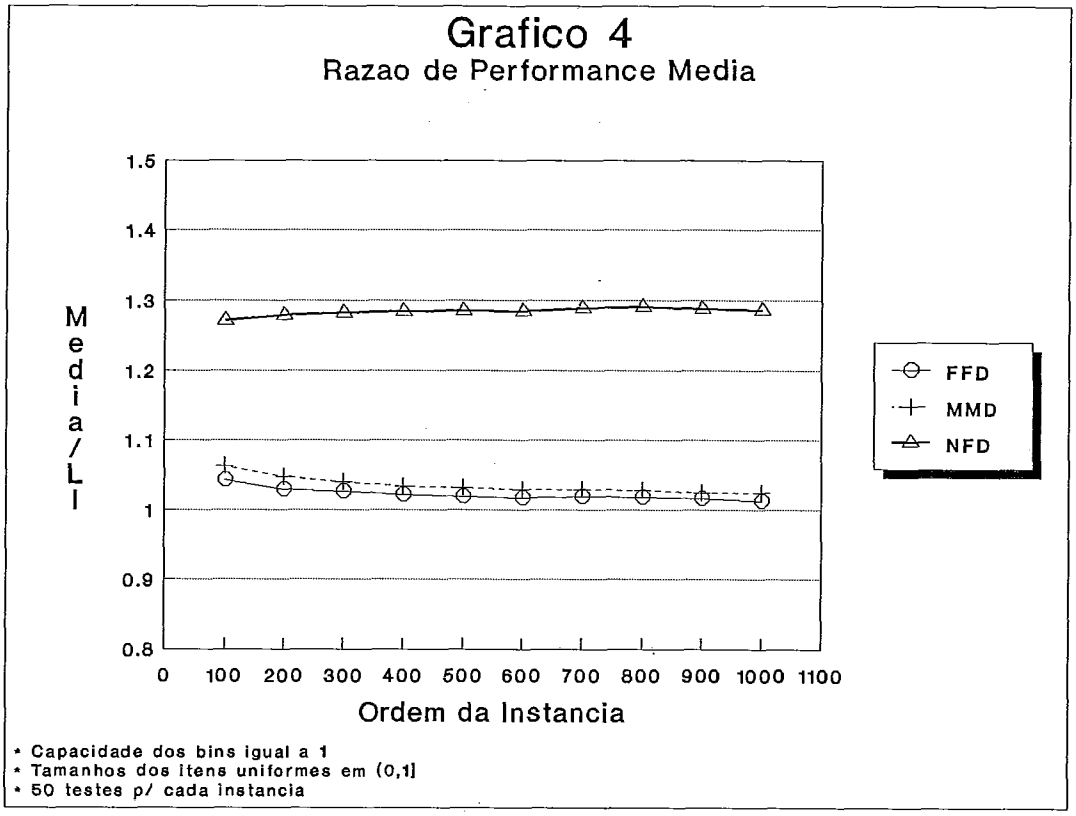


TABELA 4

Problemas com tamanhos uniformes em (0,1]
Performance Media: (Cap. dos Bins = 1)

it	n	FFD	MMD	MMDat	MMDif	MMDr1	MMDmf	NFD
50	100	1.043	1.063	1.063	1.063	1.043	1.062	1.271
	200	1.029	1.047	1.047	1.047	1.029	1.047	1.279
	300	1.026	1.040	1.040	1.040	1.026	1.040	1.282
	400	1.022	1.034	1.034	1.034	1.022	1.034	1.284
	500	1.019	1.032	1.032	1.031	1.019	1.031	1.285
	600	1.017	1.029	1.029	1.029	1.017	1.029	1.284
	700	1.019	1.029	1.029	1.029	1.019	1.029	1.289
	800	1.018	1.028	1.028	1.028	1.018	1.028	1.291
	900	1.016	1.025	1.025	1.025	1.016	1.025	1.288
	1000	1.013	1.025	1.025	1.025	1.013	1.025	1.286

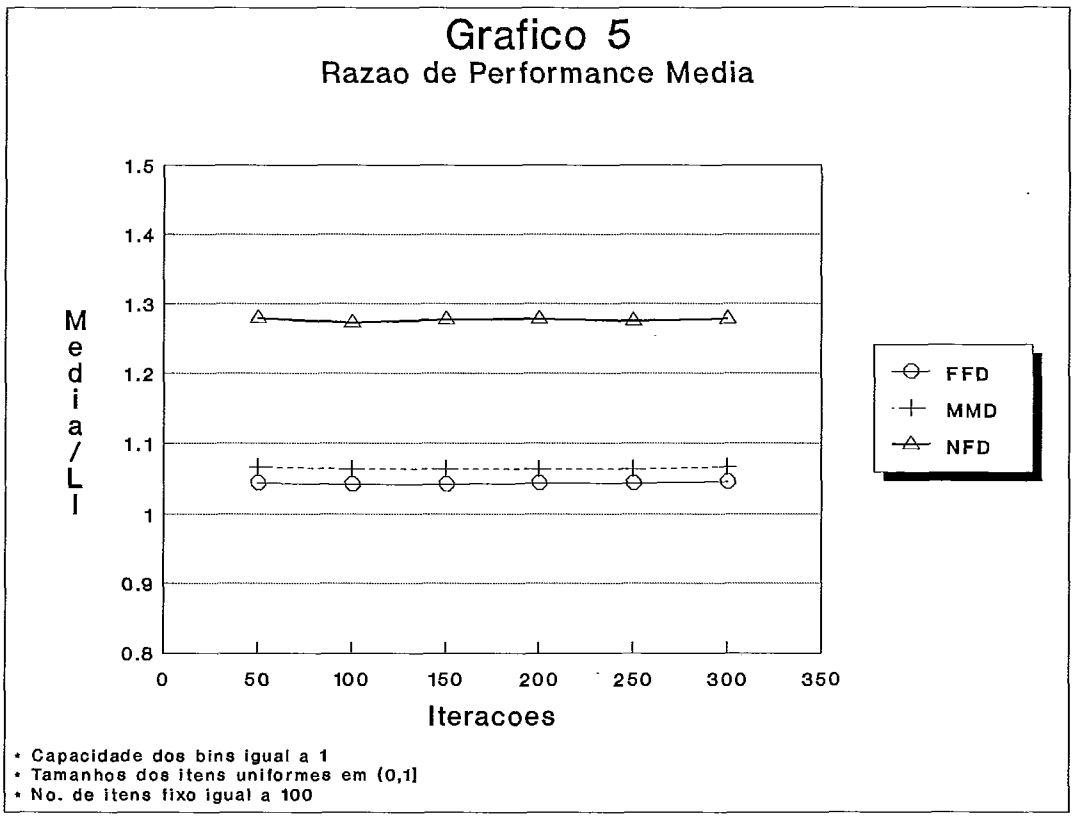


TABELA 5

Problemas com tamanhos uniformes em (0,1] Performance Media: (Cap. dos Bins = 1)								
it	n	FFD	MMD	MMDat	MMDif	MMDr1	MMDmf	NFD
50	100	1.045	1.064	1.064	1.063	1.046	1.063	1.272
100		1.041	1.064	1.064	1.063	1.041	1.063	1.274
150		1.045	1.063	1.062	1.062	1.045	1.062	1.277
200		1.043	1.065	1.065	1.064	1.043	1.064	1.277
250		1.047	1.067	1.067	1.066	1.047	1.065	1.278
300		1.045	1.064	1.064	1.063	1.045	1.063	1.279

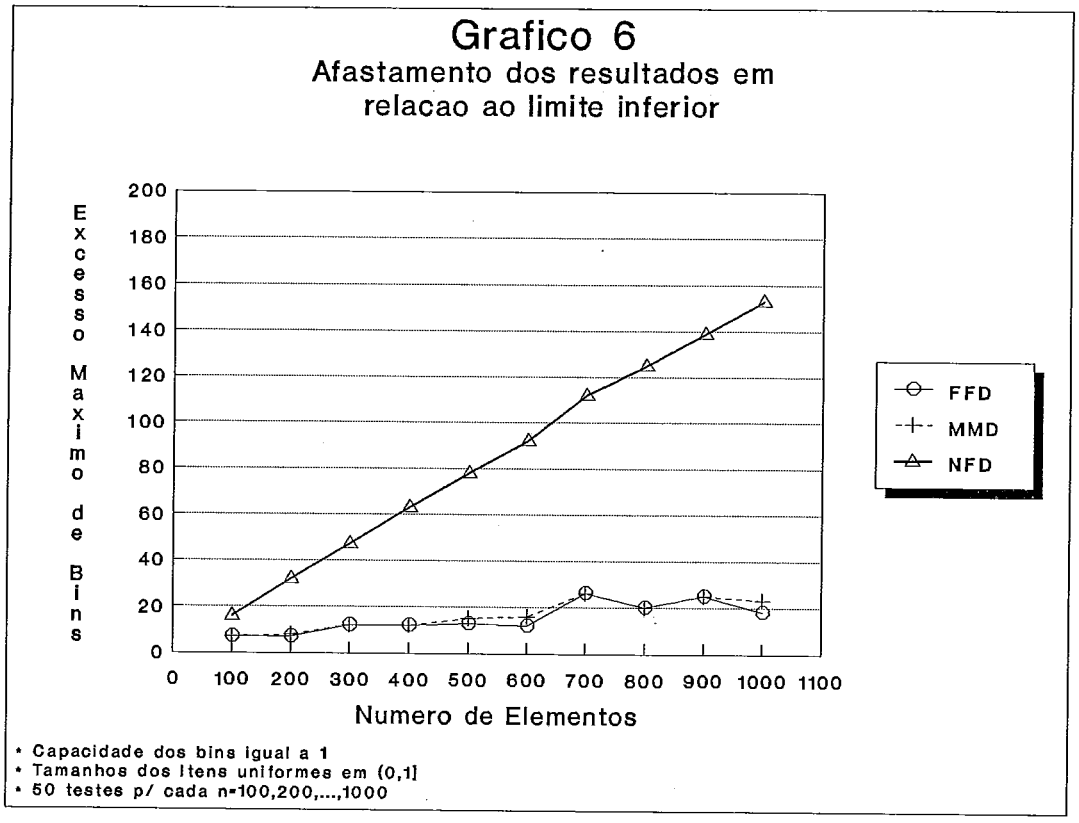


TABELA 6

Problemas com tamanhos uniformes em (0,1]
Excesso Maximo de Bins: (Cap. dos Bins = 1)

it	n	FFD	MMD	MMDat	MMDif	MMDr1	MMDmf	NFD
50	100	7	7	7	7	7	7	16
	200	7	8	8	8	7	8	32
	300	12	12	12	12	12	12	47
	400	12	12	12	12	12	12	63
	500	13	15	15	15	13	14	78
	600	12	16	16	16	12	16	92
	700	26	26	26	26	26	26	112
	800	20	20	20	20	20	20	125
	900	25	25	25	25	25	25	139
	1000	18	23	23	23	18	23	153

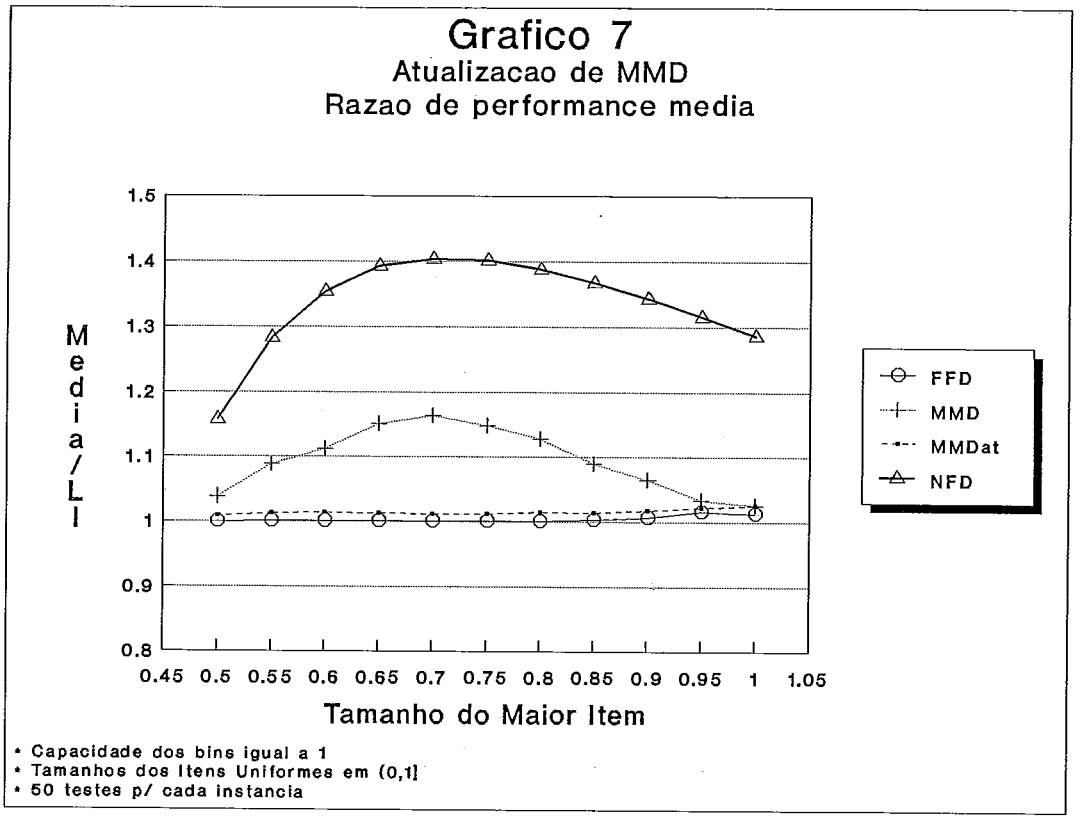


TABELA 7

Probs.c/ tamanhos uniformes em (0,1] Performance Media: (50 testes)				
Tam.	FFD	MMD	MMDat	NFD
0.5	1.000	1.038	1.009	1.158
0.55	1.001	1.088	1.013	1.283
0.6	1.001	1.112	1.014	1.354
0.65	1.001	1.151	1.013	1.393
0.7	1.001	1.164	1.012	1.404
0.75	1.001	1.148	1.012	1.402
0.8	1.001	1.127	1.015	1.388
0.85	1.003	1.089	1.014	1.368
0.9	1.007	1.065	1.018	1.343
0.95	1.016	1.033	1.021	1.316
1	1.013	1.025	1.025	1.286

Nas Tabelas 8 e 9, consideramos instâncias onde o limite inferior usado no cálculo da performance aproximada dos algoritmos é o próprio valor ótimo. Este valor, na verdade, foi previamente estabelecido, pois a priori tomamos um número fixo de bins considerados completamente cheios e através da procedure *GERADOR* as instâncias foram produzidas pelo fracionamento desses bins. Esta procedure, primeiro randomicamente seleciona o número de itens n_j a ser designados para o bin *j*, entre *2* e *dom + 1*, onde *dom* (domínio) é um parâmetro de entrada. Os tamanhos dos itens são então gerados randomicamente dividindo a capacidade do bin *j* em n_j partes (veja Fig. IV.10).

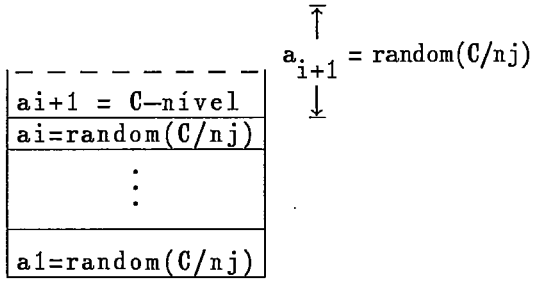


FIG. IV.10: *Particionamento de um bin*

Note que o número de itens para uma instância não é fixado, mas o número de itens esperado é *((dom + 3)/2)* × *Nº de BINS*, onde *dom* é uma faixa de valores inteiros para os quais devemos dividir aleatoriamente cada bin do empacotamento ótimo.

Os Gráficos 8 e 9, seguintes, ilustram os resultados contidos naquelas tabelas.

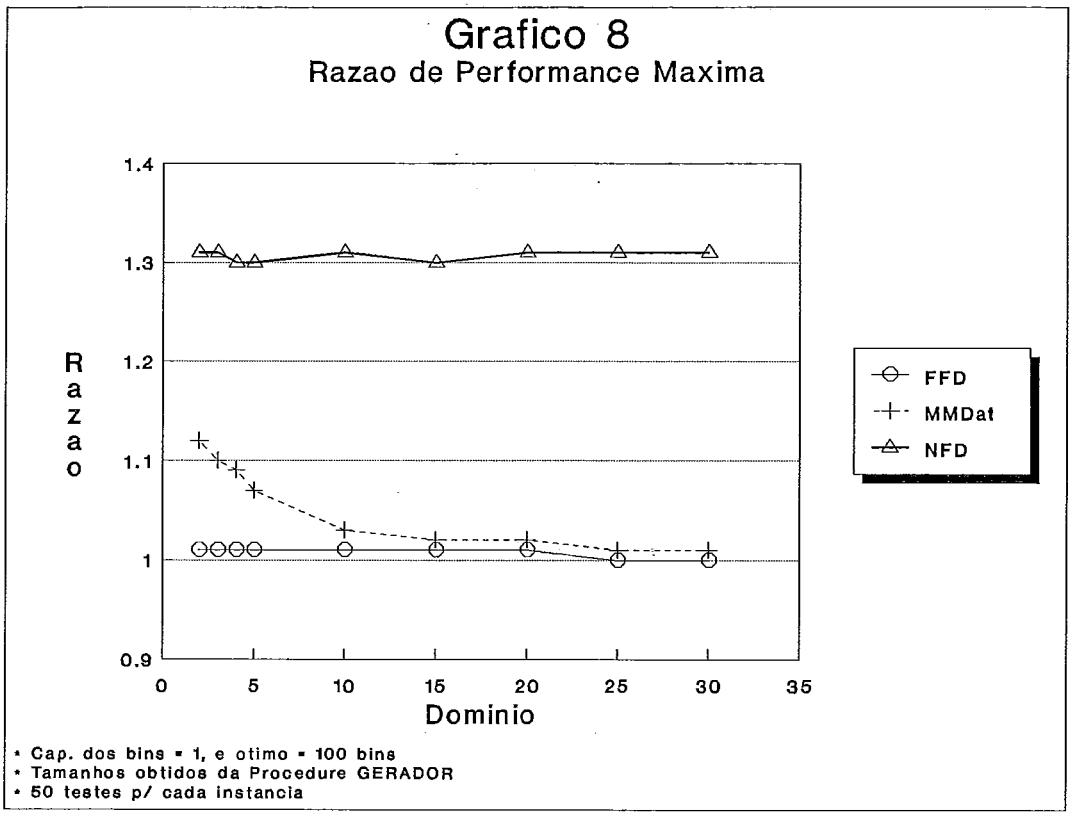


TABELA 8

Problemas com tamanhos uniformes em (0,1]
Performance Maxima: (50 testes p/cada dominio)

dom	FFD	MMD	MMDat	MMDif	MMDr1	MMDmf	NFD
2	1.010	1.120	1.120	1.080	1.010	1.030	1.310
3	1.010	1.100	1.100	1.070	1.010	1.020	1.310
4	1.010	1.090	1.090	1.060	1.010	1.020	1.300
5	1.010	1.080	1.070	1.060	1.010	1.010	1.300
10	1.010	1.050	1.030	1.040	1.010	1.010	1.310
15	1.010	1.040	1.020	1.030	1.010	1.010	1.300
20	1.010	1.030	1.020	1.020	1.010	1.010	1.310
25	1.000	1.030	1.010	1.020	1.010	1.010	1.310
30	1.000	1.020	1.010	1.020	1.010	1.010	1.310

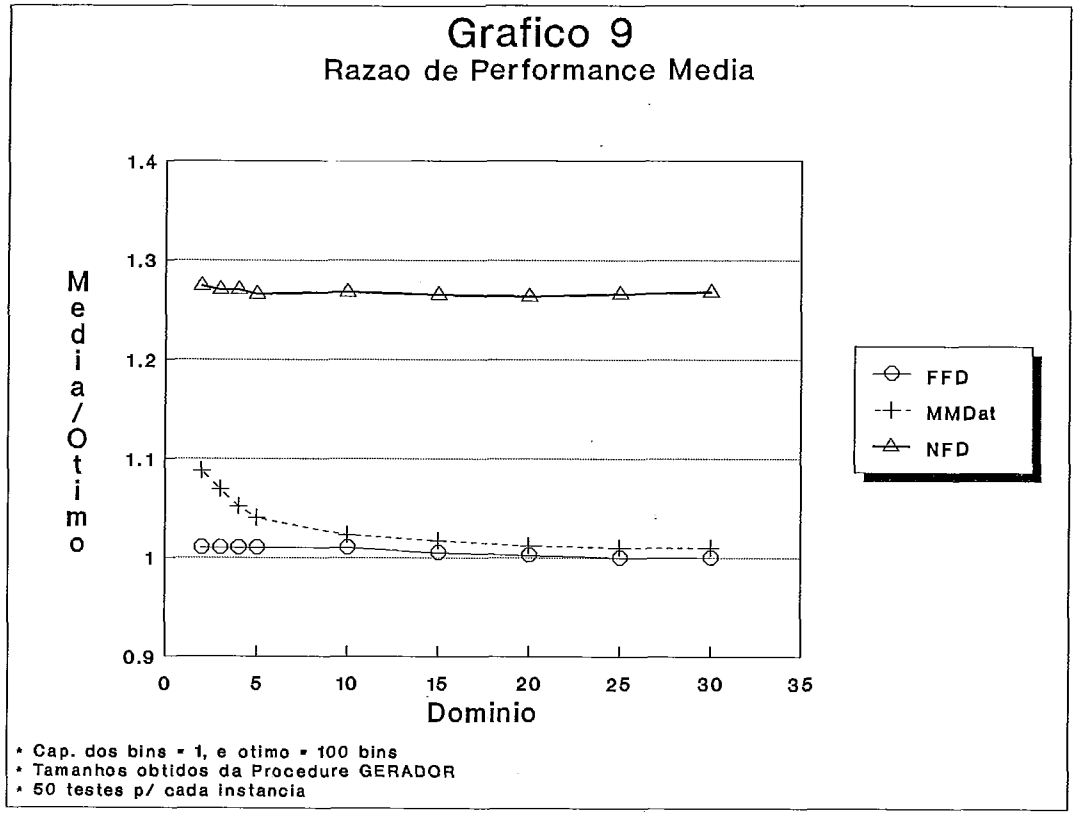


TABELA 9

Problemas com tamanhos uniformes em (0,1]
Performance Media: (50 testes p/cada dominio)

dom	FFD	MMD	MMDat	MMDif	MMDr1	MMDmf	NFD
2	1.010	1.092	1.088	1.067	1.010	1.024	1.274
3	1.010	1.091	1.069	1.058	1.010	1.015	1.270
4	1.010	1.080	1.052	1.051	1.010	1.010	1.270
5	1.010	1.067	1.040	1.046	1.010	1.010	1.266
10	1.010	1.041	1.023	1.029	1.010	1.010	1.268
15	1.005	1.031	1.017	1.021	1.010	1.010	1.265
20	1.002	1.026	1.012	1.019	1.010	1.010	1.263
25	1.000	1.020	1.010	1.015	1.010	1.010	1.266
30	1.000	1.020	1.010	1.011	1.010	1.010	1.268

Finalmente (veja Tabelas 10 e 11), apresentamos vários exemplos onde os resultados obtidos são melhores para as novas heurísticas.

TABELA 10

EXEMPLOS DE LISTAS COM RESULTADOS FAVORÁVEIS AS NOVAS HEURÍSTICAS (C = 100):

1)	57	48	26	26	23	20								
2)	57	45	29	26	26	17								
3)	73	47	31	22	15	12								
4)	67	61	22	20	10	9	4	4	3					
5)	71	64	24	12	12	9	8							
6)	70	41	33	26	15	7	6	2						
7)	61	57	15	15	13	12	2	7	4	3	1			
8)	84	41	27	20	12	9	7							
9)	60	53	15	15	14	9	9	6	6	6	5	2		
10)	74	49	46	9	6	5	5	2	2	2				
11)	66	65	26	9	9	7	6	5	5	2				
12)	39	36	34	30	11	10	10	9	7	6	5	3		
13)	53	48	17	9	9	9	8	8	8	7	7	6	6	4	1
14)	46	45	28	27	14	13	10	8	6	3				
15)	69	63	43	37	29	28	20	11						

TABELA 11

RESULTADOS CORRESPONDENTES AS LISTAS:

Problema	FFD	MMD	MMDat	MMDif	MMDr1	MMDmf	NFD	OPT
1)	3	***2***	3	***2***	3	3	3	2
2)	3	***2***	3	***2***	3	3	3	2
3)	3	***2***	3	***2***	3	3	3	2
4)	3	3	***2***	3	***2***	3	3	2
5)	3	***2***	3	***2***	3	3	3	2
6)	3	***2***	3	***2***	3	***2***	3	2
7)	3	***2***	***2***	***2***	3	***2***	3	2
8)	3	***2***	3	***2***	3	3	3	2
9)	3	3	***2***	3	***2***	3	3	2
10)	3	3	3	3	3	3	3	2
11)	3	***2***	3	***2***	3	***2***	3	2
12)	3	3	3	3	***2***	3	3	2
13)	3	***2***	3	***2***	3	***2***	3	2
14)	3	***2***	***2***	***2***	3	3	3	2
15)	4	4	4	***3***	4	4	4	3

Procedure GERADOR(N,dom,Lista);

<< Definição dos parametros de entrada:

C capacidade dos bins;
nj nº de ítens selecionados aleatoriamente para o bin j;
a[i] i–ésimo elemento da lista gerado randomicamente dividindo a capacidade C do bin por nj;
dom domínio de valores para nj; >>

BEGIN
Para cada BIN faça:
nj := random(dom) + 1;
repita:
Inc(i);
a[i] := random(C/nj) e nível := nível + a[i]
até nível > C ou nº de ítens > nj;
Se nível+a[i] > C, então faça a[i] := C – (nível);
Se nível = C, então i := i – 1;
até o N–ésimo bin;
END;

TABELA 12

Complexidade Computacional dos Algoritmos Propostos	
Algoritmo	Ordem de Complexidade
FFD	O(nlogn)
MMD	O(n)
MMDat	O(n)
MMDif	O(n)
MMDr1	O(nlogn)
MMDmf	O(nlogn)
NFD	O(n)

Obs: Sem considerar a ordenação.

A seguir mostramos alguns exemplos didáticos, encontrados na literatura. Coincidentemente, o algoritmo *MMD* apresentou um resultado superior ao de *FFD*.

PROBLEMA 1: (SYSLO [23 p.526])

Dados de entrada:

L = {12, 10, 6, 5, 3, 2, 2, 2}
C = 14

Resultado:

FFD = 4
MMD = 3
solução ótima = 3

PROBLEMA 2: (CHVATAL [3, p.211])

Dados de entrada:

L = {52(x6), 29(x6), 27(x6), 21(x12)}
C = 100

Resultado:

FFD = 11
MMD = 10
solução ótima = 9

PROBLEMA 3: (FISHER [10, p.8])

Dados de entrada:

L = {21(x6), 12(x6), 11(x6) 8(x12)}
C = 40

Resultado:

FFD = 11
MMD = 10
solução ótima = 9

PROBLEMA 4: (CHVATAL [4, p.21])

Dados de entrada de FFD:

L = {285,188(x6), 126(x18), 115(x3), 112(x3), 60, 51, 12(x3),
10(x6), 9(x12)} e C = 396

Resultado:

FFD = 13
MMD = 12
solução ótima = 9

Os exemplos acima, de certa forma, indicam que este algoritmo aproximativo também pode ser considerado um modelo para estudo e aprendizagem de como desenvolver heurísticas em Otimização Combinatória.

4.1.7 Uma aplicação prática: O programa COPYPLUS.

Um problema que, geralmente, se defronta quem trabalha com Micro–computadores é a falta de disquetes ou dificuldade para copiar um conjunto de arquivos de um diretório de um disco rígido em discos flexíveis. Normalmente, isto ocorre pelo fato da cópia ser feita sequencialmente (algoritmo NEXT FIT) arquivo por arquivo, e não visar um melhor aproveitamento do espaço do disquete. Talvez, uma das razões, seja por Next Fit ser um algoritmo *on–line* em relação aos disquetes (bins). Este procedimento, é feito com comandos do tipo COPY *.* do DOS.

O programa COPYPLUS, por nós desenvolvido, desde 1989, automatiza o processo de cópia e procura otimizar o número de disquetes usados, como também a quantidade de espaço perdida. Seu princípio de funcionamento, é o algoritmo *PD (Progressivo Decrescente)*, *on–line* para os bins, que após uma pré–ordenação seleciona os arquivos do maior para o menor, sempre procurando fazer um "best fit" (algoritmo guloso) em suas buscas. Isto é, o próximo arquivo selecionado é sempre o melhor que satisfaz o disquete (bin), nesta ordem.

É demonstrado em COFFMAN *et al.* [15], que o algoritmo **First Fit Decreasing** – *FFD* funciona sempre otimamente, se a lista L de ítens para designação nos bins, forma uma *sequência divisível*, i.e., se a sequência de elementos distintos

$$s_1 > s_2 > \ldots > s_i > s_{i+1} > \ldots$$

(o número de ítens em cada tamanho é arbitrário) é tal que para todo $i \geq 1$, s_{i+1} **exatamente** divide s_i .

Como a capacidade dos dispositivos computacionais (disquetes, disco rígido, memória dinâmica, etc) e tamanhos dos blocos são comumente restritos a potência de *2*, e o algoritmo *PD* é um sócia para *FFD*, para empacotamento dos bins de forma *on–line*; então, o programa COPYPLUS que utiliza este algoritmo, produzirá sempre empacotamentos perfeitos quando os tamanhos dos arquivos formarem uma instância divisível. Ou seja, a cópia dos arquivos, neste caso, será feita sempre dentro de um número mínimo e exato de disquetes. As Figuras IV.11, IV.12 e IV.13, mostram o

LAY–OUT de entrada, operação e saída do programa.

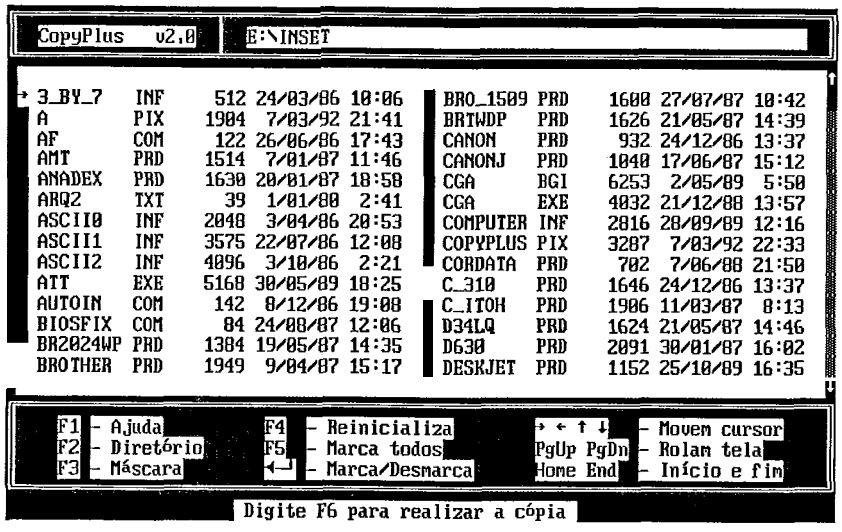


FIG. IV.11: *Tela de entrada*

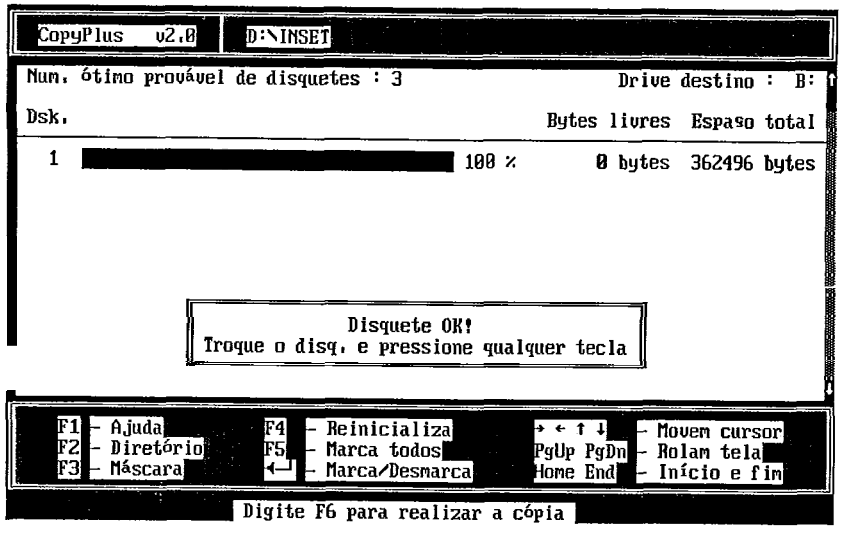


FIG. IV.12: *Tela do programa em execução*

Algumas características de COPYPLUS:

- Quase sempre utiliza o número exato de disquetes, produzindo muito frequentemente como resultado nos disquetes, com exceção do último, "0 bytes livres";
- Permite copiar todos os arquivos ou selecionar somente os desejáveis;
- Possui apresentação visual em telas gráficas.

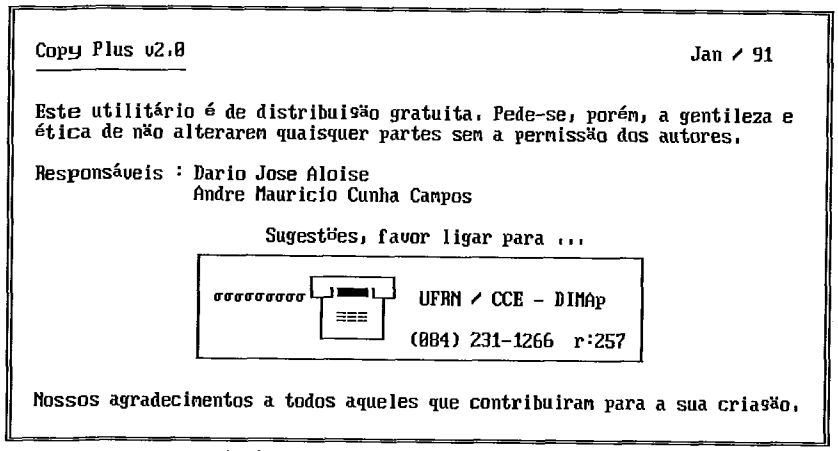


FIG. IV.13: *Tela de Saída*

Recentemente (1991), foi divulgado, num sistema Hot–Line/Rio, um programa com esta idéia de cópia automática e otimizada de arquivos, **sem** telas gráficas, idealizado por Marco Paganini.

A idéia, em si, de cópia automática e otimizada, a nosso ver, é o mais importante, porque permite a aplicação em sistemas robotizados, como numa linha de montagem, no qual cada máquina (bin) recebe um lote de instruções para operação em série. Nesta mesma implementação de COPYPLUS poderia ter sido utilizados outros algoritmos, como o Best Fit original, Worst Fit Decreasing, etc, o qual primeiro temporariamente empilha os resultados e só depois de fechar o empacotamento, fornece a saída automatizada. Preferimos a utilização do algoritmo *Progressivo Decrescente.*

4.7 O Algoritmo *IMMD* para Maximizar o número de Ítens Empacotados

Nesta variante do Bin Packing Clássico, como mencionado anteriormente, são dados um número $m \geq 1$ de bins e uma capacidade C para cada bin (por conveniência igual a *1*); o problema é empacotar tantos ítens de L quanto possível dentro de tais bins.

O conhecido algoritmo First Fit Increasing (*FFI*) trabalha em tempo *O(nlogn)*, aplicando First Fit (*FF*) a seqüência $a_1 \leq a_2 \leq \ldots \leq a_n$, e pode numa ocasião

produzir empacotamentos que contém 3/4 do número ótimo de ítens (veja COFFMAN et al [19]).

O algoritmo Iterativo Maiores e Menores Decrescente (*IMMD*): *Primeiro ordena os* ítens *de L tal que* $a_1 \geq a_2 \geq \ldots \geq a_n$ *e define listas* L_i, $1 \leq i \leq n$, *onde* L_i *é obtido de L deletando o primeiro (maior) i–ésimo* ítem, *e então executa MMD nas listas* L_1, L_2, *etc., até que uma lista seja encontrada tal que MMD empacote dentro de m ou menos bins, sendo o número de* ítens *empacotados a saída.*

A Figura IV.14 mostra um exemplo de empacotamento produzido pelos dois algoritmos.

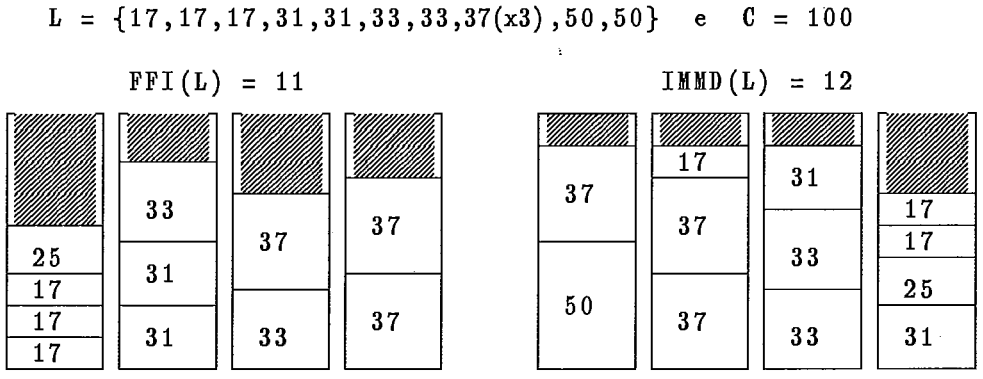


FIG. IV.14: *Exemplo de IMMD(L) > FFI(L);* $\Sigma a_i > 4$ *bins e* $OPT(L) = 12$.

Proposição 4: *Seja* $n_{IMMD}(L,m)$ *o número de* ítens *que o algoritmo IMMD empacota L dentro de m bins. Então para todo L e* $m \geq 1$, $n_{IMMD}(L,m) \geq n_{FFI}(L,m)$.

Prova: Considere que o empacotamento por *FFI* seja $B_1,\ldots,B_m$ de um prefixo L' de uma lista L. Seja $B'_1 = B_{m-i+1}$ $(1 \leq i \leq m)$; i.e., $B'_1,\ldots,B'_m$ é o reverso da sequência de bins empacotados por *FFI*, i.e., é o empacotamento feito por *NFD*. Seja $B''_1,\ldots,B''_m$ o empacotamento por *MMD* de L' dentro de m bins. Pela Proposição 3, mostramos que este empacotamento é tal que $p \in B'_i$ implica $p \in B''_j$ para algum $j \leq i$. Assim *IMMD* deve empacotar um subconjunto próprio de L' em L.

Proposição 5: *O algoritmo IMMD requer não mais do que m iterações; a complexidade de tempo no pior–caso de IMMD é, portanto, O(m·n).*

Prova: O tempo requerido por uma iteração do algoritmo *IMMD* é, devido a *MMD*, excluindo ordenação, *O(n)*. Assim, para a determinação da complexidade de tempo do algoritmo é preciso sabermos apenas quantas iterações o algoritmo necessita, no pior–caso, para empacotar dentro de *m* bins.

Suponha que $L = \{a_1, a_2, \ldots, a_n\}$ seja uma lista de ítens tal que o empacotamento por *IMMD* requeira mais do que *m* iterações, para safisfazer os *m* bins. Seja *t* um índice tal que a lista $L' = \{a_1, a_2, \ldots, a_t\}$ seja um prefixo máximo da lista *L*, isto é, $\Sigma_{i=1}^{t} a_i \leq m$. Da definição de *IMMD*, o algoritmo deve começar sua iteração inicial com *L'*, e sucessivamente ir descartando os maiores ítens até que os ítens restantes satisfaçam um empacotamento por *MMD* de não mais do que *m* bins. Assim, na *m*–ésima iteração por *MMD*, os ítens $a_t, a_{t-1}, \ldots, a_{t-(m-1)+1}$ devem ter sidos descartados. Entretanto, mantida a suposição para *L*, deve existir um ítem a_r de *L'* que não satisfaz o *m*–ésimo bin de *MMD*, e consequentemente, nenhum dos *m* primeiros bins de *MMD*. Com isso, os níveis dos *m* primeiros bins excedem $1-a_r$. Portanto, temos

$$\Sigma_{i=1}^{t-m+1} a_i > m(1 - a_r) + a_r \quad \text{ou} \quad \Sigma_{i=1}^{t-m+1} a_i > m - (m-1)a_r$$

Desde que $a_i \geq a_r$ para $[t - (m-1) + 1] \leq i \leq t$ (pois, para aplicar *IMMD* a lista precisa estar ordenada decrescentemente), temos

$$\sum_{i=1}^{t} a_i = \sum_{i=1}^{t-m+1} a_i + \sum_{i=t-m+2}^{t} a_i > m - (m-1)a_r + (m-1)a_r$$

Uma contradição ! , com o fato que $\sum_{i=1}^{t} a_i \leq m$ ■

Os Gráficos 10 e 11, correspondentes às Tabelas 13 e 14, ilustram o desempenho do algoritmo *IMMD* em relação ao algoritmo *FFI* para 50 testes realizados, em cada domínio, sabendo–se à priori da solução ótima.

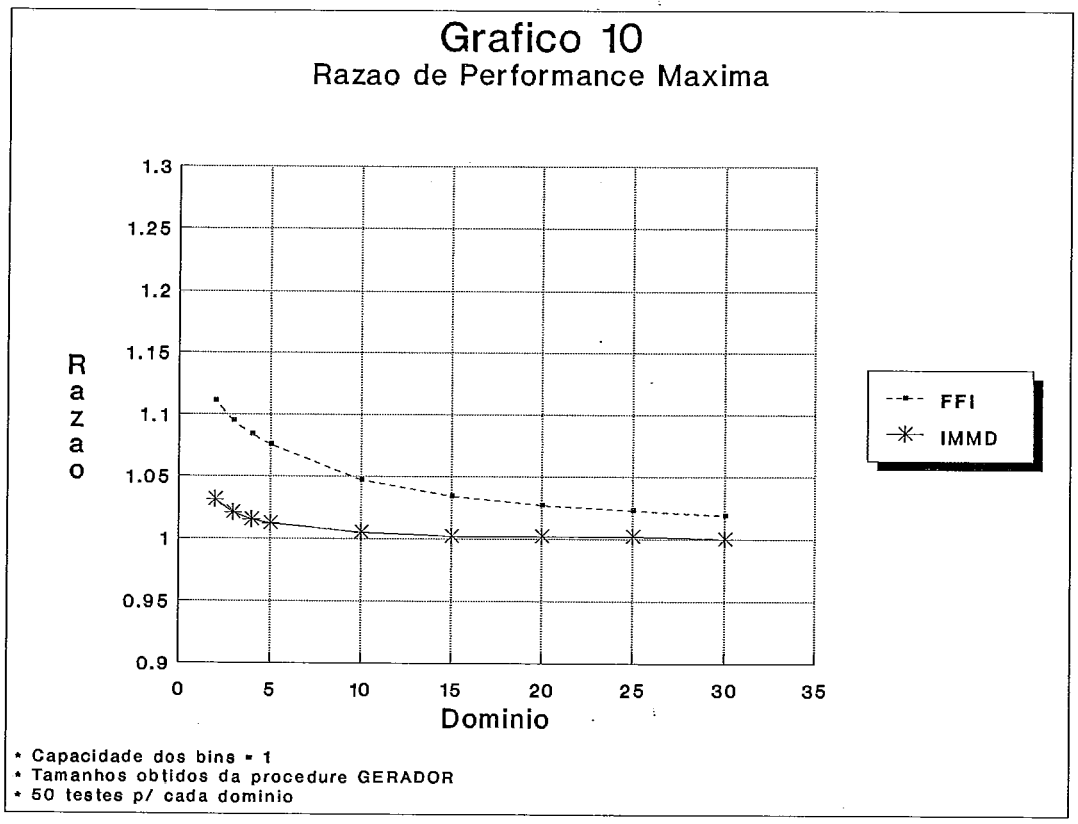


TABELA 13

Problemas com tamanhos uniformes em (0,1]
Performance Maxima (Sobra de Itens):
50 testes p/ cada dominio

dom	FFI	IMMD
2	1.111(27)	1.031(7)
3	1.095(28)	1.021(6)
4	1.084(29)	1.015(5)
5	1.076(24)	1.012(4)
10	1.047(28)	1.005(2)
15	1.034(26)	1.002(2)
20	1.027(29)	1.002(2)
25	1.023(29)	1.002(2)
30	1.019(24)	1.001(1)

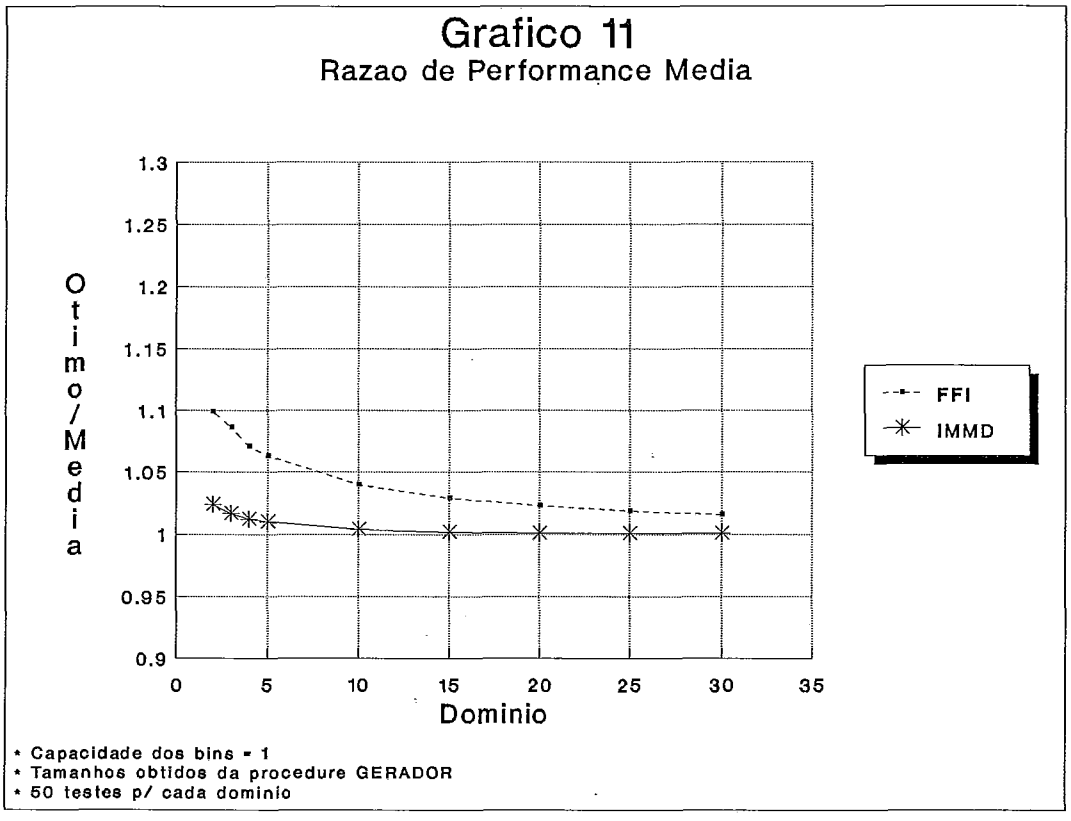


TABELA 14

Problemas com tamanhos uniformes em (0,1] Performance Media (Iter. do Algoritmo): 50 testes p/ cada dominio		
dom	FFI	IMMD
2	1.099(1)	1.024(8)
3	1.086(1)	1.017(7)
4	1.071(1)	1.012(6)
5	1.063(1)	1.010(5)
10	1.040(1)	1.004(4)
15	1.029(1)	1.002(3)
20	1.023(1)	1.001(3)
25	1.019(1)	1.001(3)
30	1.016(1)	1.001(2)

5.8 Detalhes de Implementação dos Algoritmos Propostos

(a) Algoritmos MMD, MMDat e MMdif.

Estes algoritmos foram implementados com o uso de vetores, e de uma tabela HASHING, para verificar a existencia de itens intermediários que eliminavam por completo a folga, com a seguinte declaração:

```
TYPE HASH = Record
              IndL,          {índice em HASHING }
              Freq = Integer {frequência do item}
            end;
TABELA = Array [1..MAX] of HASH;
```

Esta estrutura de dados permite um acesso direto ao item procurado, sem alterar a ordem de complexidade final do algoritmo MMD. Isto é, o acesso é *O(1)* para cada item encontrado. Por isso, quando um pré–processamento é pressuposto, MMDif mantém a complexidade em *O(n)*.

Exemplo: Seja a seguinte lista de itens L = {8,7,6,5,5,4,2,1,1,1}. Então, a tabela HASH armazena os valores na forma abaixo:

	IndL	Freq
1	8	3
2	7	1
3	0	0
4	6	1
5	4	2
6	3	1
7	2	1
8	1	1

Quando o item intermediário que completa o bin é achado, por este processo, sua frequência (*Freq*) é diminuida de 1, e caso continue com *Freq > 1* seu índice na lista (*IndL*) é aumentado de 1. Assim, se o item 5, na posição 6 da Lista, é encontrado, sua atualização na **tabela hash** seria

	antes			*depois*	
5	6	2	5	7	1

indicando que somente *um* 5 na pos(7) se encontra na lista. (Mais detalhes sobre **tabela hash**, ver *e.g.*, NYHOFF & LEESTMA [64,p.364] e HOROWITZ & SAHNI [36]).

(b) Os demais algoritmos

Foram construídos tendo por base a *Árvore B–Tree–m–way*. A mecânica de utilização foi a seguinte:

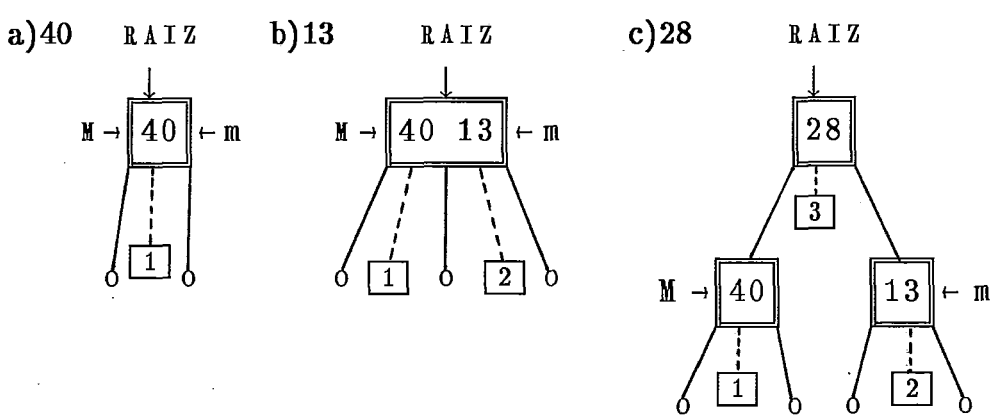
A lista de itens é alocada na *B–tree–m–way*, onde são armazenados, em cada nó, *m–1* dados. Também é armazenada a ordem de entrada na *B–Tree*, como referência na futura alocação nos "BINS".

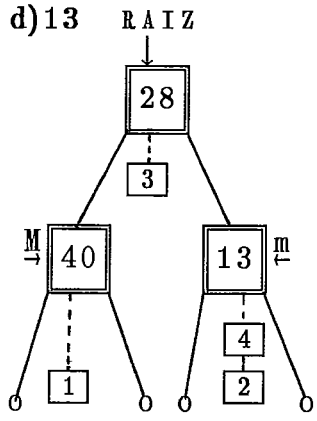
Fase 1 – Não há pré–ordenação dos dados de entrada na lista, contudo, eles são armazenados de forma decrescente, tendo como base de ordenação os valores dos dados. Cada item de entrada é composto de duas informações: o valor do item e a ordem de entrada na árvore. As colisões, ou seja, valores iguais são armazenados numa pilha simples (LIFO), onde ocorrerem.

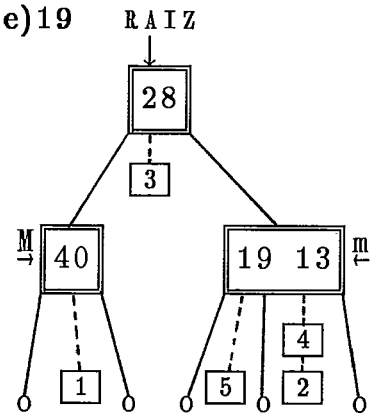
Exemplo: Dados de entrada

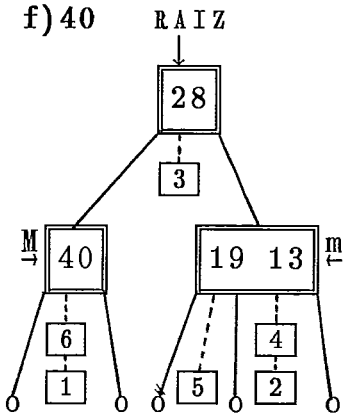
L = {40,13,28,13,19,40,13,14,5,45} e m=3 (B–Tree–3–way).

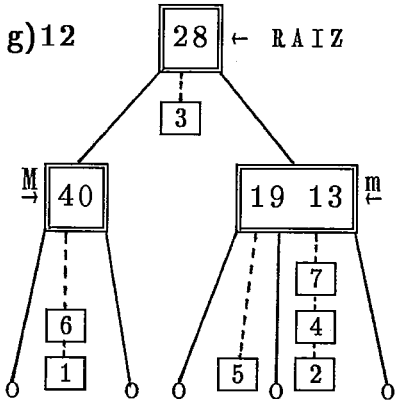
Os dados são armazenados na árvore, de acordo com a seguinte sequência de ilustrações:

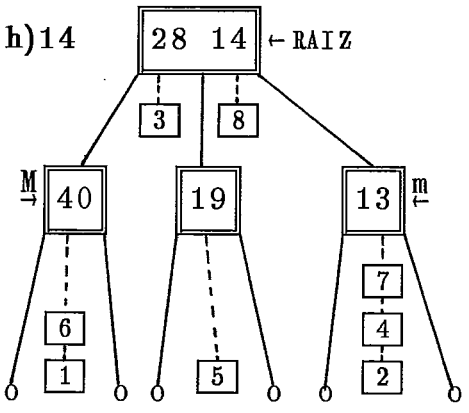


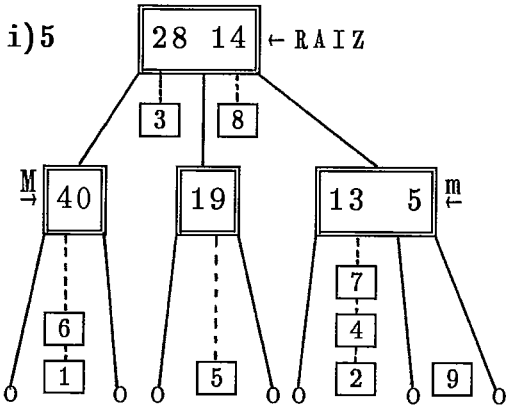


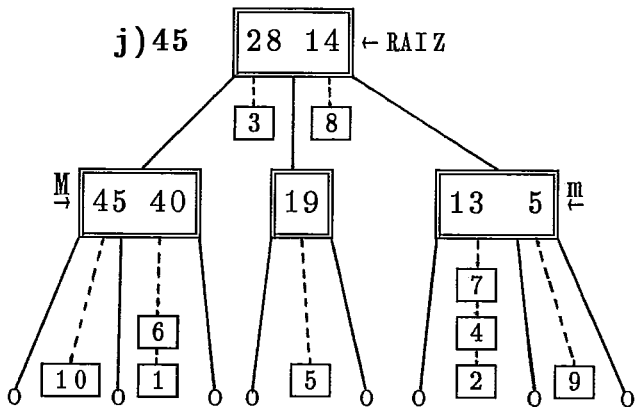










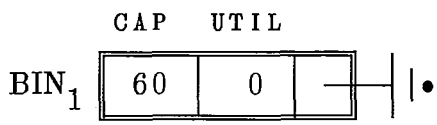


Observando a árvore final, notamos que o apontador MAIOR indica o nodo onde está o maior valor (10ª entrada) e que apontador Menor indica o nodo que contém o MENOR valor (9ª entrada). O valor 13 apareceu três 3 vezes, na 7ª, 4ª e 2ª entradas. Como se vê, as colisões são identificadas por uma lista (LIFO) simples. Existem três possíveis acessos a esta árvore para fins de remoção de seus elementos: 1) pela raiz, para acessar

o elemento intermediário; 2) pelo apontador MAIOR, para ter acesso pelo maior elemento da lista; e 3) pelo apontador MENOR, para ter acesso pelo menor valor. $O(n\log n)$ é a complexidade de entrada.

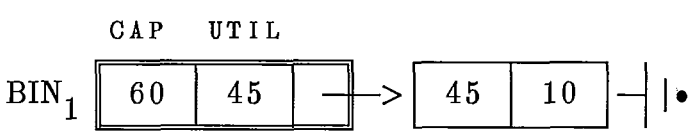
Vamos agora considerar, que os "BINS" tenham todos capacidade 60. Então, o algoritmo *PD* (Progressivo Decrescente) funciona da seguinte maneira:

1a iteração

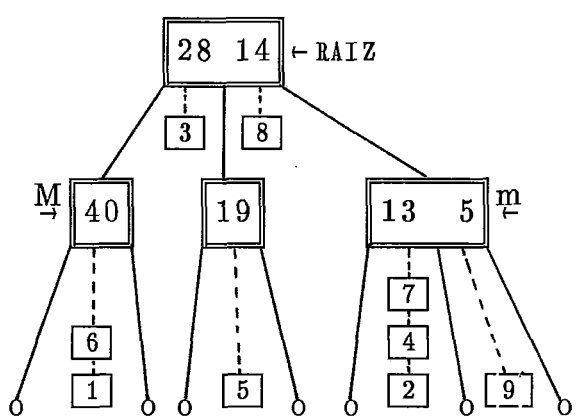


Na fase inicial, a CAPacidade do bin é igual a 60 e a UTILização é nenhuma. A FOLGA = CAP − UTIL. Iniciamos pelo apontador MAIOR, recuperamos o elemento (45,10) = (**Valor,ordem de entrada original**), porque FOLGA ≥ Valor do item.

Atualização:



Situação na árvore:



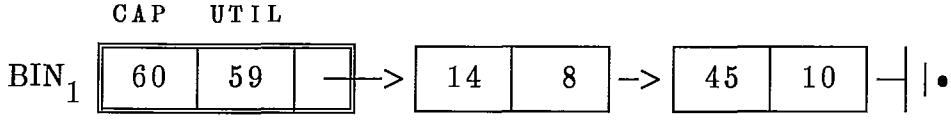
FOLGA = CAP − UTIL

- Tentamos o próximo item: Valor = 40 ≥ CAP − UTIL = 15.
- Verificamos o menor valor ⇒ 5 < CAP − UTIL {Entretanto, é possível que exista um ítem de valor melhor}.
- Acessando pela RAIZ, achar o elemento intermediário i, de valor V_i, tal que:

$$\max_i V_i$$
$$\text{s.a.: } 0 \leq V_i \leq \text{FOLGA}$$

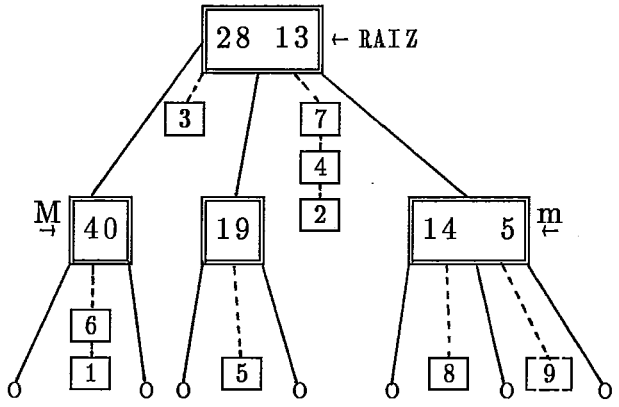
Acessando pela RAIZ $V_i = 14$ uma solução viável, porém pode existir uma melhor anterior, a qual deverá estar na sub–árvore à esquerda. Na sub–árvore, $V_i = 19 >$ FOLGA, portanto, não é viável. Assim, usando "back–tracking", $V_i^* = 14$. Como resultado, temos o par (14,8) como o melhor elemento (best–fit) intermediário satisfazendo ao "BIN_1".

Atualização:

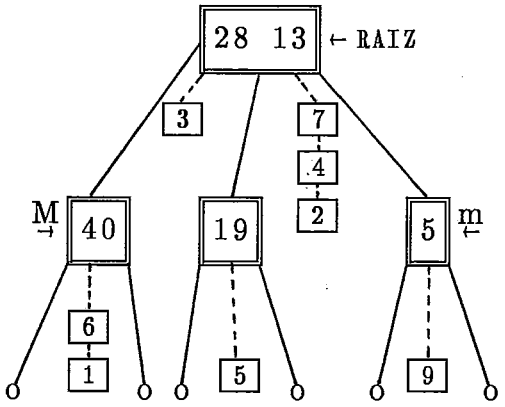


Para remover o elemento (14,8), o algoritmo da árvore **B–tree–3–way** permuta com outro valor que seja uma folha. Assim, antes de remover, teremos na árvore o que segue:

Situação na árvore:



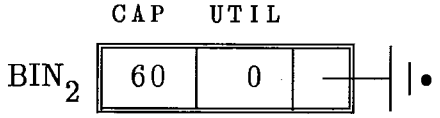
Remove–se (14,8) e teremos:



FOLGA = 60 − 59 = 1. Como FOLGA = 1 e MENOR Valor = 5, então o BIN_1 é fechado.

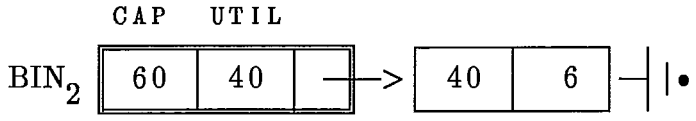
2a iteração

Temos:



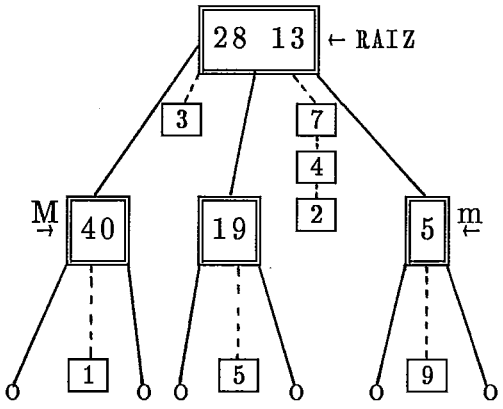
Iniciamos pelo apontador MAIOR. Recuperamos o elemento (40,6)

Atualização



Como 40 tem frequencia > 1, seu valor é comparado com a FOLGA. Como não satisfaz, visto que FOLGA = 20, testamos o próximo elemento cujo valor = 28 (isto é obtido através do nó_pai de 40).

Situação na árvore



- Tentamos o próximo item: Valor = 28 ≥ CAP − UTIL = 20.
- Verificamos o menor valor ⇏ 5 < CAP − UTIL {Entretanto, é possível que exista um ítem de valor melhor}.
- Acessando pela RAIZ, achamos o elemento intermediário i, de valor V_i, tal que:

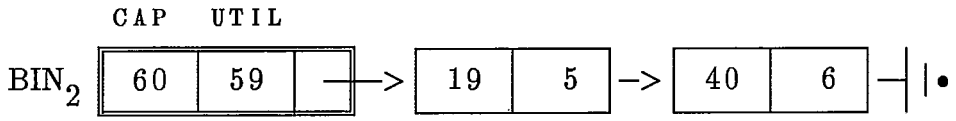
$$\max_i V_i$$

$$\text{s.a.: } 0 \leq V_i \leq \text{FOLGA}$$

Temos que, acessando pela RAIZ, $v = 13$ é uma solução viável, porém pode existir uma

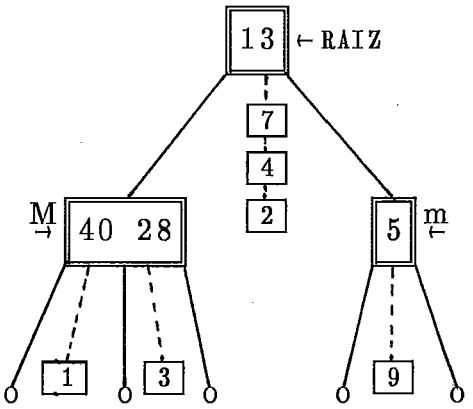
melhor anterior, a qual deverá estar na sub–árvore à esquerda de 13; achamos $V_i = 19$. Como este nó é uma folha e de valor < FOLGA, então $v_i^* = 19$. Como resultado, temos o par (19,5) como o melhor elemento (best–fit) intermediário satisfazendo ao "BIN_2".

Atualização



• Removendo–se o elemento (19,5) a árvore se reduz por "sibling" (ver algoritmo do HOROWITZ & SHANI [35,p.591]) a:

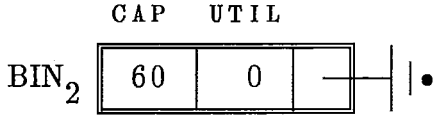
Situação na árvore ("sibling")



Como FOLGA = 60 − 59 = 1 < valor = 5, então o BIN_2 é fechado.

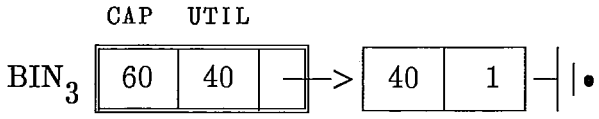
3a iteração

Temos:

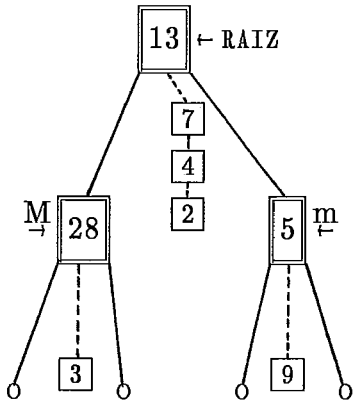


• Iniciamos pelo apontador MAIOR. Recuperamos o elemento (40,1)

Atualização

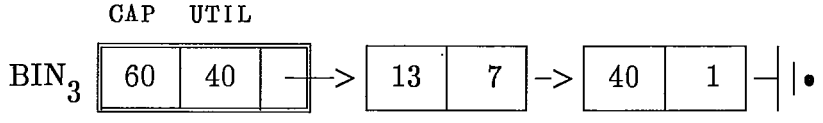


Situação na árvore

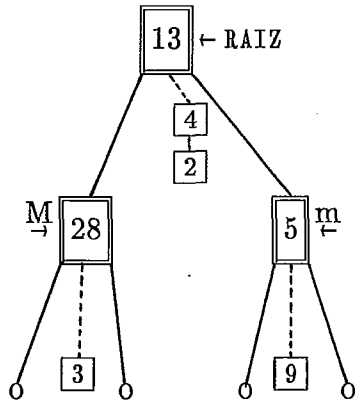


Como 28 > FOLGA, pela RAIZ achamos $v_i^* = 13$. O par (13,7) é o "best fit" intermediário.

Atualização



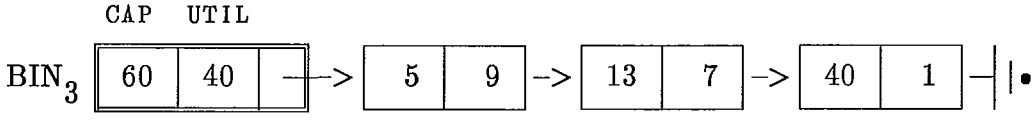
Situação na árvore



• Acessando o próximo elemento de valor = 13, isto é, (13,4) não satisfaz, contudo testamos o próximo elemento ≠ 13.

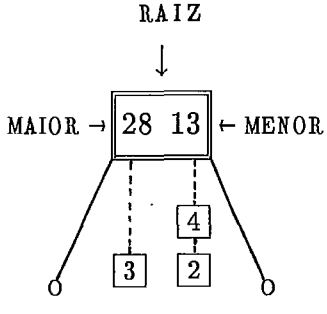
• (5,9) é obtido e 5 < FOLGA.

Atualização



• Eliminando–se o elemento (5,9) haverá novo "sibiling".

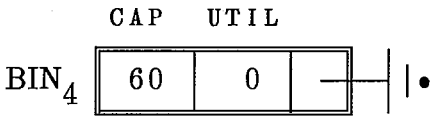
Situação na árvore



• Como os apontadores MAIOR, MENOR e RAIZ apontam para o mesmo NÓ o problema se reduz a uma busca numa lista indexada. Como FOLGA < 13 (menor valor) o BIN_3 é fechado

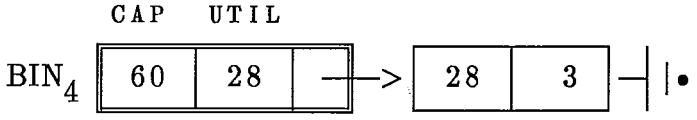
4a iteração

Temos:

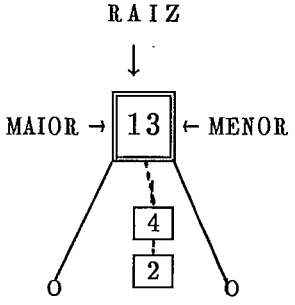


• Iniciamos pelo apontador MAIOR. Recuperamos o elemento (28,3).

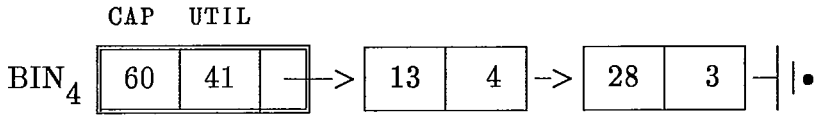
Atualização



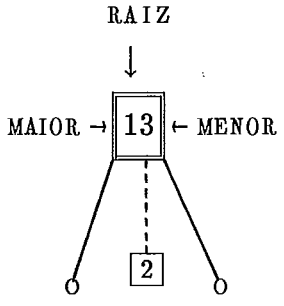
Situação na árvore

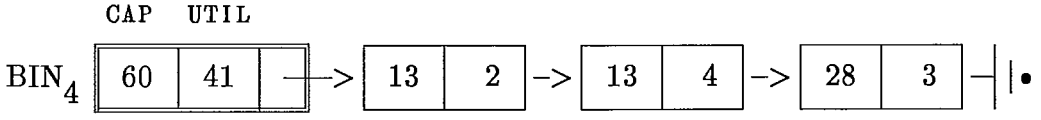


Atualização



situação final na árvore





• Em seguida, árvore vazia ! FIM do algoritmo.

A Figura IV.15 exibe o resultado final do empacotamento do algoritmo *PD* usando árvore *B–Tree–3–way*.

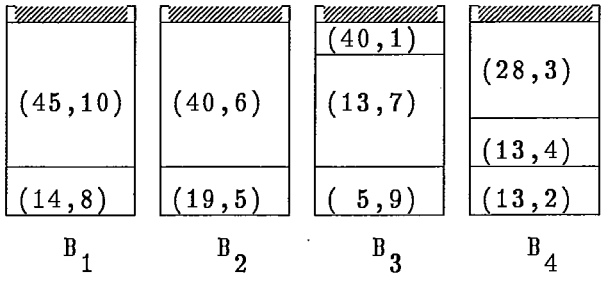


FIG. IV.15: *Saída do algoritmo PD através de árvore B–Tree–3–way*

Obs: No programa, a pilha dos elementos colocados nos BIN's não têm o valor armazenado, apenas o índice de ordem da lista de entrada, onde eles podem ser recuperados.

Capítulo V
Algoritmos Aproximativos multi–dimensionais

5.1 O Problema Bin–Packing Multidimensional

O problema de Bin–Packing é, freqüentemente, generalizado para dimensões maiores de duas maneiras: através de **vector packing** (empacotamento de vetores) e empacotamento de hiper–paralelepípedos. Em cada caso o tamanho do i–ésimo ítem é especificado por um vetor $(a_{i1},a_{i2},\ldots,a_{id})$ de números reais entre 0 e 1. Uma instância L de um problema de empacotamento d–dimensional é especificada por um vetor $(a_{ij})_{nxd}$ cuja i–ésima linha especifica o tamanho do i–ésimo ítem. Para qualquer um desses empacotamentos de ítens dentro de bins, pode ser estabelecida uma medida de *perda*, representando o *espaço total não usado* nos bins. Se L_m denotar o conjunto de ítens empacotados no m–ésimo bin, então, no caso de empacotamento de vetores, a medida de perda é $\Sigma_m \Sigma_{j=1}^{d}(1 - \Sigma_{i\epsilon L_m} a_{ij})$. No caso de empacotamento de paralelepípedos a medida apropriada é

$$\Sigma_m(1 - \Sigma_{i\epsilon L_m} \prod_{j=1}^{d} a_{ij})$$

Nos dois casos, considera–se que um empacotamento que miniminize perda também minimizará o número de bins usado. Sejam $Perda_V(L)$ a perda de um empacotamento ótimo de vetores para uma instância L, e $Perda_R(L)$ para o caso de hiper–paralelepípedos. Então, se A_V e A_P são os algoritmos aproximativos para estes casos, respectivamente; segue que para qualquer instância L

$$A_V(L) \geq Perda_V(L) \quad \text{e} \quad A_P(L) \geq Perda_P(L).$$

5.1.1 O Empacotamento de vetores

No *vector packing*, a condição que um dado conjunto $S \subseteq \{1,2,...,n\}$ de ítens possa ser empacotado dentro de um bin é que, para $j = 1,2,...,d$, $\sum_{i \in S} a_{ij} \leq 1$. Isto é, a capacidade de cada bin é também um vetor d–dimensional (1,1,...,1), e o objetivo é empacotar os ítens num número mínimo de bins, dado que o conteúdo de qualquer bin deve ter uma **soma vetorial** menor do que ou igual a sua capacidade. Uma interpretação usada é que as d coordenadas do vetor representam diferentes recursos, tais como *tempo*, *espaço*, *energia* e *custo*, e a_{ij} representa a quantidade de recurso j consumida pelo ítem i. É requerido que os objetos empacotados conjuntamente num bin não possam coletivamente consumir mais do que uma unidade de qualquer recurso. Este problema modela scheduling de multiprocessadores de tarefas de comprimentos unitários no caso quando existem d recursos, ao invés de um único como no caso uni–dimensional.

A Figura V.1, mostra uma versão 2–dimensional do problema. Um vetor (a_{i1},a_{i2}) poderia ser pensado como representando um retângulo com comprimento a_{i1} e largura a_{i2}, e um bin de capacidade (1,1) como um quadrado dentro do qual os retângulos são empacotados.

GAREY, GRAHAM, JOHNSON, e YAO [30] analizaram o problema d–dimensional e fizeram adaptações dos algoritmos *FF* e *FFD* para este caso (em *FFD* os ítens são ordenados em ordem não–crescente pela componente de tamanho máxima dos vetores). Eles mostraram que $R^{\infty}_{FF} = d + 7/10$, o qual reduz–se ao resultado familiar *17/10* do caso uni–dimensional, e que $d \leq R^{\infty}_{FF} \leq d + 1/3$. YAO [70] mostrou que qualquer algoritmo que tenha um tempo de execução que é $O(n\log n)$ na árvore de decisão, deve ter $R^{\infty}_{A} \geq d$.

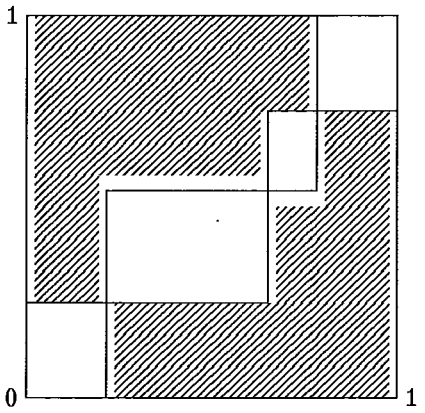


FIG. V.1: *Bi–dimensional vetor packing*

5.1.2 O empacotamento de hiper–paralelepípedos

O problema de empacotar hiper–paralelepípedos tem um sentido mais geométrico que o anterior. O *i*–ésimo ítem representa uma **caixa** *d*–dimensional cujos lados são $a_{i1}, a_{i2}, \ldots, a_{id}$ e pode ser representado como um ponto no $(0,1]^d$, e cada **bin** é uma **caixa** cujos lados, para simplificar, são de comprimento unitário (i.e., um hipercubo). Os ítens alocados a uma dada caixa devem ser acomodados de forma viável, isto é, sem insterseção. Existem diversas variações do problema de empacotar paralelepípedos; dentre essas a permissão de rotação e deslocamento de posição dos ítens dentro do seu próprio bin.

Este problema para $d = 2$ e 3 dimensões, possuem diversas aplicações em Ciência da Computação e Pesquisa Operacional. Por exemplo, alocação de memória em sistemas multiprocessadores onde tarefas (jobs) dividem uma memória comum. Neste caso, os ítens retangulares representam tarefas com espaço e requerimento de tempo conhecidos, a dimensão horizontal é a quantidade de memória contígua e a vertical o tempo. Aos ítens não são permitidos rotação devido a restrição de tempo.

Um exemplo de empacotamento 3–D é a geração de padrões de **pallets**, p. ex., quando carregando um grande container de um navio, ou quando empilhando caixas em caminhões.

5.2 O Problema Bin Packing Bi–Dimensional

Existem, na literatura (cf. COFFMAN *et al.* [14]) essencialmente dois tipos de problemas de bin packing 2–D bastante pesquisados. A saber,

Dado *uma coleção ou lista* $L = \{r_1, r_2, \ldots, r_n\}$ *de retângulos*:

(1) ***Strip Packing:*** *empacotar os elementos de L dentro de uma faixa F retangular aberta em uma das extremidades ou não, de forma a minimizar a extensão da faixa necessária. Os lados dos retângulos menores (*ítens*) devem ser paralelos aos lados de F, e quaisquer dois* ítens *retangulares não podem ser sobrepostos. A Rotação nos* ítens *de* 90^0 *pode ser permita ou não;*

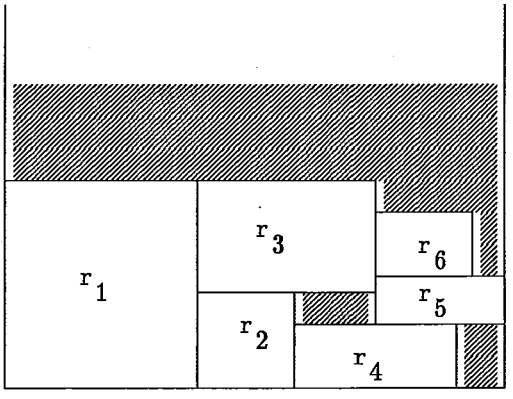


FIG. V.2: *Um strip packing.*

(2) ***Natural 2–D Bin Packing:*** *empacotar os elementos de L dentro de um número mínimo de bins retangulares idênticos, R, tal que os lados dos retângulos menores sejam paralelos aos lados de R, e quaisquer dois* ítens *não possam se sobrepor;*

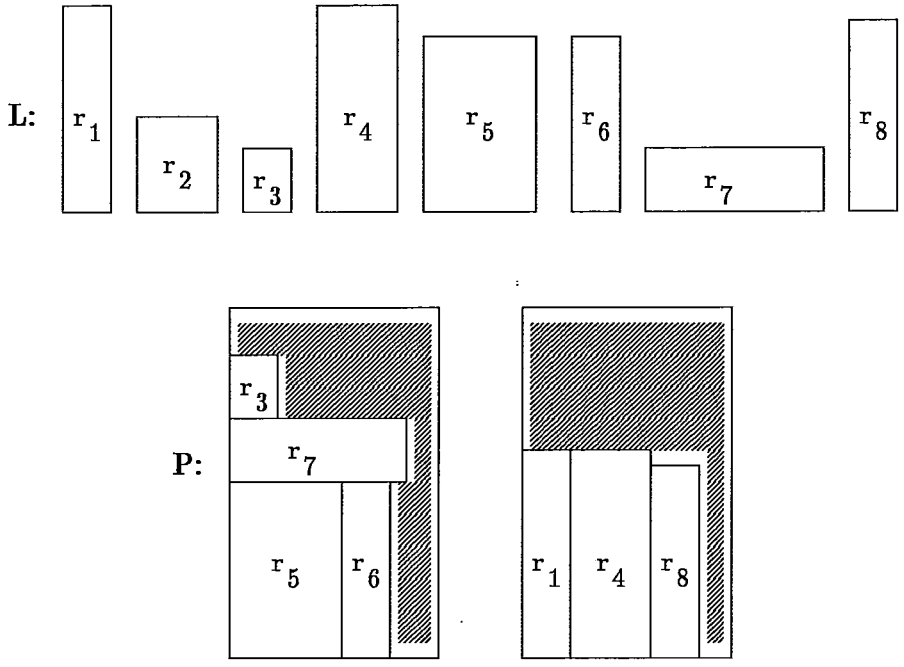


FIG. V.3: *Exemplo de um empacotamento P em 2 bins.*

5.2.1 Algoritmos aproximativos *off–line* 2–D e performance

Duas regras básicas de empacotamento têm sido empregadas na construção de algoritmos para estes dois problemas: "BOTTOM UP – left justified" ou "BOTTOM–LEFT", e "LEVEL ORIENTED" (i.e., a qual obtém níveis sem permitir rotação. Do contrário, diz–se só "LEVEL"). Os algoritmos que fazem uso de qualquer uma dessas regras empacotam os ítens na sequência em que estes se apresentam originalmente na lista L.

Na regra BOTTOM–UP os ítens (retângulos) são empacotados, em sequência, justificados à esquerda tão próximo à base da faixa quanto possa satisfazer e tão distante da margem esquerda quanto ele possa ser colocado naquele nível. Na regra LEVEL–ORIENTED, cada nível no empacotamento corresponde a um bin e a altura do empacotamento corresponde ao número de bins usado. Os ítens retangulares são colocados com sua linha base num nível e tão longe da esquerda quanto possível no bin. O primeiro nível num bin é a base do bin. Cadà bin subsequente é definido pela altura do ítem mais alto colocado no nível anterior (veja Figura V.3).

Assim, a *altura de um nível* é definida como a distância da base da faixa à base do bin. O primeiro nível tendo altura 0 (zero).

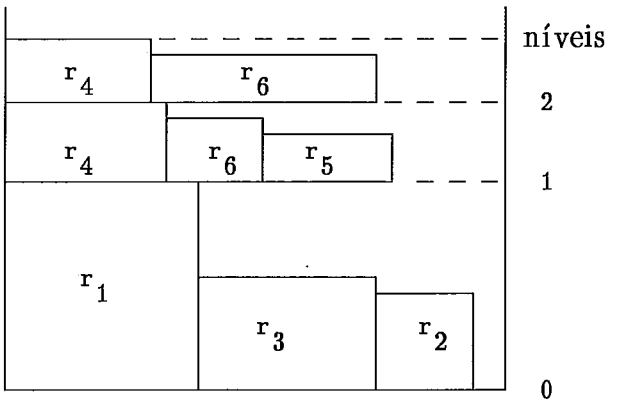


FIG. V.3: *Um exemplo para LEVEL ORIENTED*

(a) Algoritmos BOTTOM–LEFT

Em BAKER, COFFMAN & RIVEST [4], uma variedade de algoritmos aproximativos foram desenvolvidos e analisados quanto à performance, obedecendo a regra BOTTOM–LEFT, de acordo com a ordenação inicial dos ítens. Assim, usando a notação abreviada de Bl para o algoritmo BOTTOM–LEFT sem ordenação nos ítens, e BLIW, BLIH, BLDW e BLDH para os algoritmos com pré–ordenação por: largura crescente (increasing width), altura crescente, largura decrescente e altura decrescente, respectivamente, então foi estabelecido a razão de performance absoluta para cada um destes algoritmos: $R_{BL} = R_{BLIW} = R_{BLIH} = R_{BLDH} = \infty$. Ou seja, são arbitrariamente ruins, e $R_{BLDW} = 3$. Para o caso particular de empacotamentos de quadrados, este limite foi melhorado para $R_{BLDW} = 2$. D. J. BROWN [7], construiu instâncias na qual o melhor empacotamento por BOTTOM–LEFT possível produziu uma faixa cuja altura é 5/4 da ótima.

Para o caso de retângulos arbitrários, o algoritmo clássico FFDH (veja Figura V.3), o qual usa a regra de "LEVEL" (COFFMAN, GAREY, JOHNSON & TARJAN [16]), para empacotamento numa faixa vertical sem topo, apresenta uma performance absoluta de $R_{FFDH} = 2.7$ e o algoritmo de SLEATOR [67], reduziu este limite para 2.5 .

Embora os resultados dos algoritmos acima tenham sido obtidos para razão de

performance absoluta considerando que a altura da faixa vertical possa ser definida como um valor arbitrariamente grande, algumas aplicações práticas podem impor um limite superior nesta altura. Neste caso, a análise assintótica fornece um resultado menor e pode ser uma medida mais significativa. Por exemplo, $R^{\infty}_{BLDW} = 2$, uma melhoria de 1 sobre a performance absoluta, porém longe do ótimo.

(b) Algoritmos de Nível ("level" algorithms)

A procura por algoritmos com melhor performance assintótica foi primeiro realizada por COFFMAN *et al.* [16]. Dois algoritmos básicos foram propostos: O NEXT FIT DECREASING HEIGHT (NFDH) e o FIRST FIT DECREASING HEIGHT (FFDH). Estes algoritmos são baseados nas idéias dos seus análogos algoritmos uni–dimensionais. No algoritmo NFDH os retângulos são empacotados justificados à esquerda num nível até que o próximo retângulo não satisfaça, este então passa a iniciar um novo nível acima do anterior, e no qual o empacotamento é continuado. No algoritmo FFDH, cada retângulo é colocado justificado à esquerda no primeiro (i.e., mais baixo) nível no qual satisfaça. Curiosamente, a razão de performace no pior–caso, foi exatamente como no caso uni–dimensional para NF e FF : $R^{\infty}_{NFDH} = 2$ e $R^{\infty}_{FFDH} = 1.7$. Para o caso de quadrados o limite é reduzido para 1.5 .

Gerou–se, então, uma questão se haveria algum algoritmo de nível que atingisse o limite de 11/9 de FFD, no caso bi–dimensional. COFFMAN *et al.* [16], apresentaram o algoritmo SPLIT FIT com $R^{\infty}_{SF} = 1.5$. Sua estratégia consiste em particionar o conjunto de retângulos em duas partes, uma delas **com** larguras dos retângulos excedendo C/2 e a outra **sem**, e cada subconjunto é ordenado por altura não–crescente. O empacotamento se dá pela combinação dos dois subconjuntos. Esta idéia de particionar dentro de subconjuntos de acordo com a largura dos retângulos foi aproveitado no trabalho de BAKER, BROWN & KATSEFF [3], os quais apresentaram um algoritmo mais complicado com $R^{\infty}_{A} \leq 5/4$, limite este muito próximo de $11/9$.

Para o **natural 2–D bin–packing**, do nosso conhecimento, somente o algoritmo *FFDH*FFD* de CHUNG, GAREY & JOHNSON [8] foi analisado sob o ponto de vista de performance do pior–caso. A idéia do algoritmo é como segue: Suponha que um bin tenha largura W e altura H. Primeiro use *FFDH* para empacotar o conjunto de retângulos dentro de uma faixa de largura W. Em seguida, decomponha este empacotamento dentro de blocos correspondentes aos níveis criados por *FFDH*. Cada bloco pode ser visto como um retângulo de largura W e altura igual a altura do nível. Portanto, empacotando estes blocos dentro de bins retangulares de largura W torna–se um problema simples de bin–packing uni–dimensional, onde o tamanho de um ítem (bloco) é sua altura. Aplique *FFD* para este problema uni–dimensional (Observe Figura V.4).

A performance assintótica deste algoritmo foi estimada em

$$2.022 \leq R^{\infty}_{FFDH^*FFDH} \leq 2.125\ldots .$$

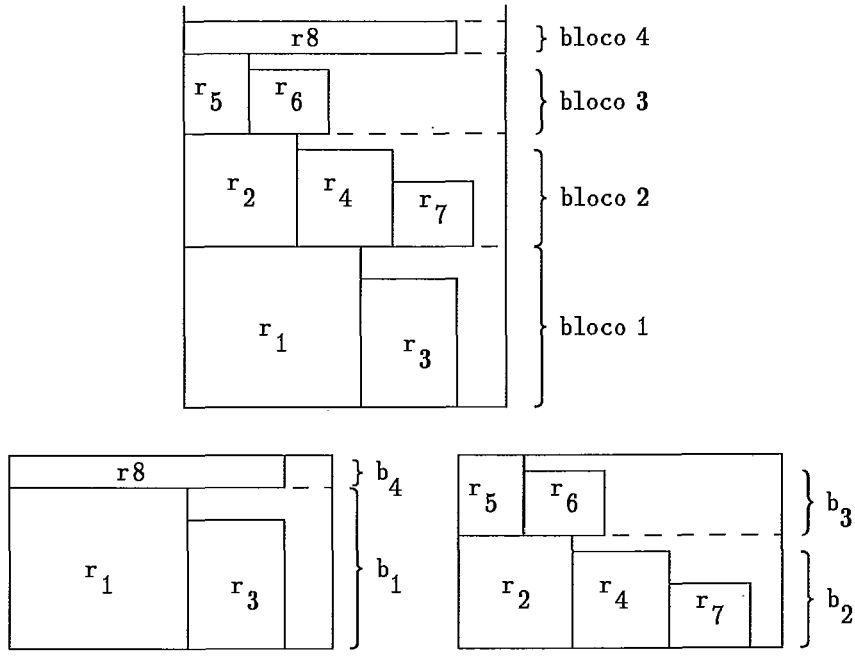


FIG. V.4: *Um exemplo de FFDH*FFD*

5.2.2 O Algoritmo MMD Bi–Dimensional

O algoritmo BI–MMD sugerido é uma extensão natural do algoritmo MMD, para situações bi–dimensionais, utilizando que a idéia de compensação para minimizar a perda, começando com ítens maiores e depois menores. Analisando a literatura especializada no assunto de empacotamentos Bi–Dimensionais deve ser observado que este algoritmo aproveita a idéia do algoritmo de COFFMAN & LAGARIAS [18,1989], usado para empacotamento de quadrados numa Faixa (strip) vertical. Naquele trabalho, a aplicação da fase decrescente e depois crescente é feita procurando o encaixe dos ítens, e objetivando minimizar a altura da Faixa. Este mesmo raciocínio tem sido aplicado por COFFMAN & SHOR [12,1990] para o caso mais geral de empacotamentos de retângulos. Em ambos, é feita apenas a análise do caso–médio dos algoritmos apresentados. Os algoritmos para o natural 2–D bin–packing são desenvolvidos destes primeiros, através da aplicação da regra de níveis.

No nosso estudo, a aplicação de Maiores (em ordem decrescente) e Menores (em ordem crescente) é realizada sobre a dimensão da altura fixa e a dimensão horizontal livre. A idéia de encaixe do algoritmo A de COFFMAN & LAGARIAS [18], é então aplicada em retângulos genéricos formando, diferentemente daquele algoritmo, várias pilhas paralelas consecutivas que tendem a se encaixarem, e no qual determinam o comprimento horizontal que se deseja minimizar. O tempo de execução de *BI–MMD* é dominado pelo tempo $O(n\log n)$ da ordenação dos retângulos. As duas situações são mostradas na Fig. V.5. Note, na primeira ilustração, que se a largura da faixa fosse variável, poderíamos deminuí–la imprensando as duas pilhas formadas pelas etapas das bases crescentes e decrescentes, respectiva e consecutivamente. Este raciocínio é tranposto para a outra ilustração à direita.

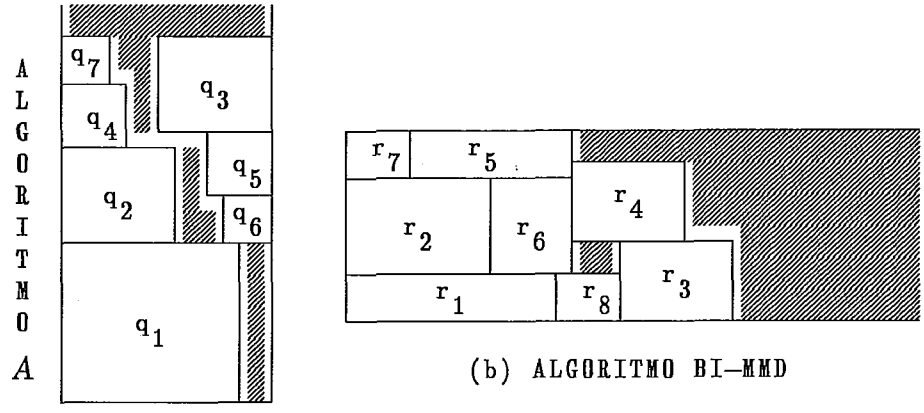


FIG. V.5: *Um exemplo de strip packing dos algoritmos A e BI–MMD.*

O Algoritmo BI–MMD:

(1) Empilhe as caixas (retângulos) maiores de L, BOTTOM–LEFT em ordem *decrescente* de largura, ao longo da margem esquerda ou contorno resultante, enquanto satisfizer altura da faixa.

(2) Empilhe as caixas pequenas de L, BOTTOM–LEFT em ordem *crescente* de largura, ao longo da margem esquerda ou contorno resultante, enquanto satisfizer altura da faixa.

(3) O Algoritmo pára quando não existir mais caixas na lista L para serem alocadas. Caso contrário, repita sucessivamente os passos (1) e (2).

Como um procedimento computacional:

```
Procedure BI–MMD;
< Definição de Tipos e Variáveis >
begin
        < Pré–ordenação nas bases dos retângulos e Inicialização >
        Repeat
          Ative_Primeira_Caixa_da_Lista;
          While (Existe folga suficiente na altura da Pilha) And (Lista ≠ ∅) Do
                Use Progressivo Decrescente;
          End_While;
          Ative_Ultima_Caixa_da_Lista;
          While (Existe folga suficiente na altura da Pilha) And (Lista ≠ ∅) Do
                Use Progressivo Crescente;
          End_While;
        Until (Lista = ∅);
end;
```

Se a faixa é limitada também na horizontal, por digamos x, então temos a mudança:

```
Repeat
  Ative_Primeira_Caixa_da_Lista;
  While (Existe folga suficiente na altura da Pilha)
        And (Lista ≠ ∅) Do
              If (Não ultrapassou limite x) Then
                    Use Progressivo Decrescente;
  End_While;
  Ative_Ultima_Caixa_da_Lista;
  While (Existe folga suficiente na altura da Pilha)
        And (Lista ≠ ∅) Do
              If (Não ultrapassou limite x) Then
                    Use Progressivo Crescente;
  End_While;
Until (Lista = ∅) Or (limite x ultrapassado);
```

A justificação das caixas é feita por um **procedimento** que procura por um grupo de caixas, numa mesma pilha mais próxima, que passam pelo intervalo dos níveis inferior e superior (y_i,y_s), determinado pela caixa que se quer justificar, selecionando–se, então, aquela caixa mais à direita (i.e., a de maior abscissa x). Veja Figura V.6.

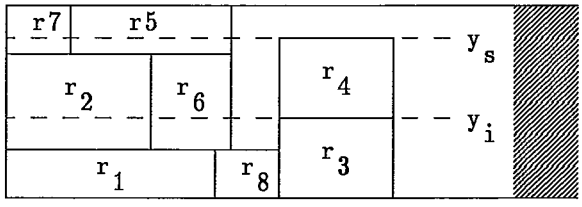


FIG. V.6: *A justificação de uma caixa.*

Nesta figura, as caixas r_2 e r_6 se encontram dentro dos níveis (y_i,y_s), mas somente a caixa r_6 está na pilha mais próxima e é, portanto, através dela (a de maior abscissa x), que a caixa r_4 é justificada,

A rotação de 90 graus nos ítens é permitida e realizada pela duplicação da lista de dados, invertendo–se as coordenadas x e y dos novos ítens fictícios.

A performance dos algoritmo é feita experimentalmente, ao invés de analiticamente. Um conjunto de 100 caixas foram geradas randomicamente, para testes, fazendo $x = y \leftarrow round(random(F/2))+5$, onde x e y são, respectivamente, as coordenadas horizontal e vertical das caixas, e F é a largura da faixa tomada com valor igual a 100. A Tabela 13 exibe os valores gerados, o qual é apenas uma amostra.

TABELA 13

LISTA BI-BIDIMENSIONAL GERADA ALEATORIAMENTE PARA TESTE (LARG. FAIXA = 100):

(5 11)	(34 31)	(41 22)	(40 8)	(50 51)	(29 21)	(29 11)	(19 26)	(20 16)	(23 45)
(27 28)	(8 51)	(12 48)	(46 19)	(16 41)	(28 31)	(5 33)	(48 48)	(33 46)	(21 35)
(49 9)	(25 25)	(13 37)	(11 46)	(49 11)	(41 7)	(40 8)	(21 13)	(34 31)	(18 33)
(33 19)	(13 37)	(26 28)	(23 36)	(52 53)	(30 15)	(32 28)	(11 19)	(28 48)	(24 24)
(33 18)	(42 21)	(6 42)	(21 8)	(35 21)	(32 48)	(26 34)	(5 52)	(45 24)	(16 31)
(21 20)	(7 22)	(36 39)	(19 13)	(14 33)	(5 24)	(21 20)	(50 21)	(32 6)	(6 13)
(7 13)	(33 32)	(34 14)	(21 41)	(24 23)	(33 37)	(25 34)	(28 26)	(33 29)	(52 11)
(25 36)	(8 38)	(45 12)	(20 12)	(7 25)	(37 40)	(22 6)	(24 29)	(36 20)	(9 33)
(26 24)	(35 54)	(40 43)	(53 39)	(39 11)	(19 30)	(37 49)	(28 7)	(15 37)	(32 6)
(8 7)	(40 23)	(44 13)	(32 26)	(34 51)	(15 37)	(32 6)	(50 50)	(51 48)	(50 31)

A Tabela 14 mostra os resultados obtidos para as duas situações do algoritmo, **sem** e **com** rotação das caixas, quando o número de caixas é gradativamente aumentado de 10 em 10 caixas até o limite de 100. O Gráfico 12 ilustra o desempenho dos algoritmos.

A Tabela 15 e Gráfico 13, correspondente, mostram o desempenho dos algoritmos quando a largura da Faixa sofre uma variação de $\pm$ 40 e o número de caixas é fixado em 100.

A notação empregada foi:

AREAT : área total das caixas

AREAC : área da faixa consumida para empacotamento das caixas

REND = (AREAT/AREAC)* 100%

PERDA = (1 − REND)*100%

DIST : distância atingida pelo empacotamento na faixa

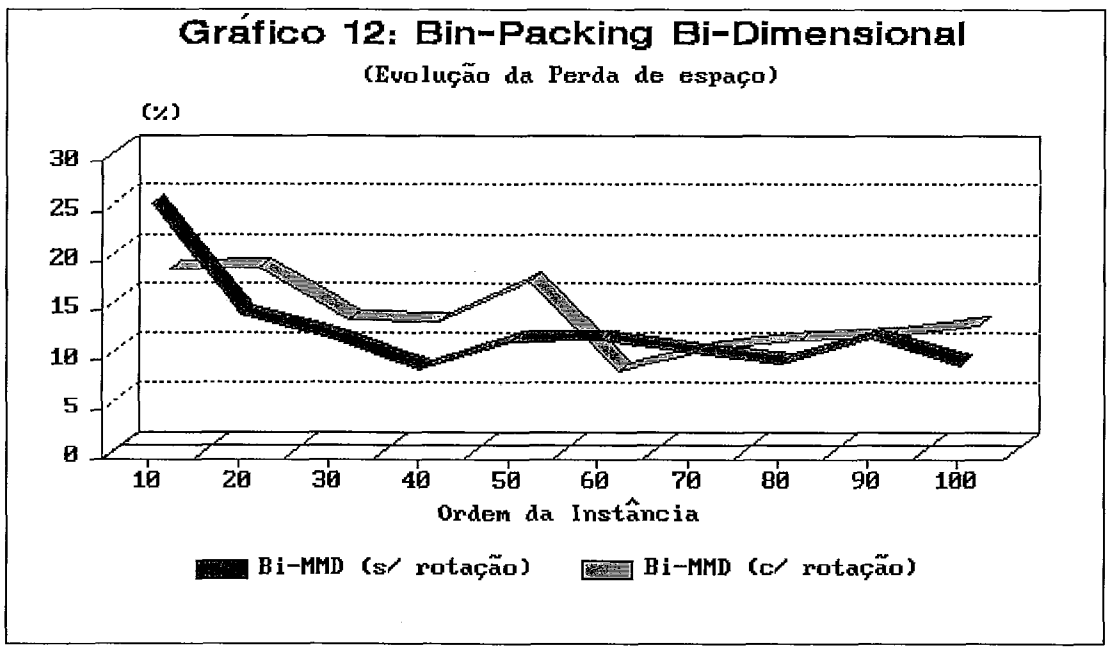


TABELA 14

RESULTADOS CORRESPONDENTES A LISTA DE ENTRADA (LARG. FAIXA = 100):

ORDEM	BI-MMD (sem ROTACAO) AREAT	AREAC	REND	PERDA	BI-MMD (com ROTACAO) AREAT	AREAC	REND	PERDA
10	7658	10300	74,35%	25,65%	-	9300	82,34%	17,66%
20	16518	19300	85,59%	14,41%	-	20100	82,18%	17,82%
30	21638	24600	87,96%	12,04%	-	24700	87,60%	12,40%
40	30533	33500	91,14%	8,86%	-	34800	87,74%	12,26%
50	37420	42400	88,25%	11,75%	-	44700	83,71%	16,29%
60	41967	47600	88,17%	11,83%	-	45300	92,64%	7,36%
70	49331	55200	89,37%	10,63%	-	54500	90,52%	9,48%
80	54815	60600	90,45%	9,55%	-	61200	89,57%	10,43%
90	64871	73800	87,90%	12,10%	-	72800	89,11%	10,89%
100	76230	84100	90,64%	9,36%	-	86600	88,03%	11,97%
MEDIA	40098	45140	87,38%	12,62%	-	45400	87,34%	12,66%

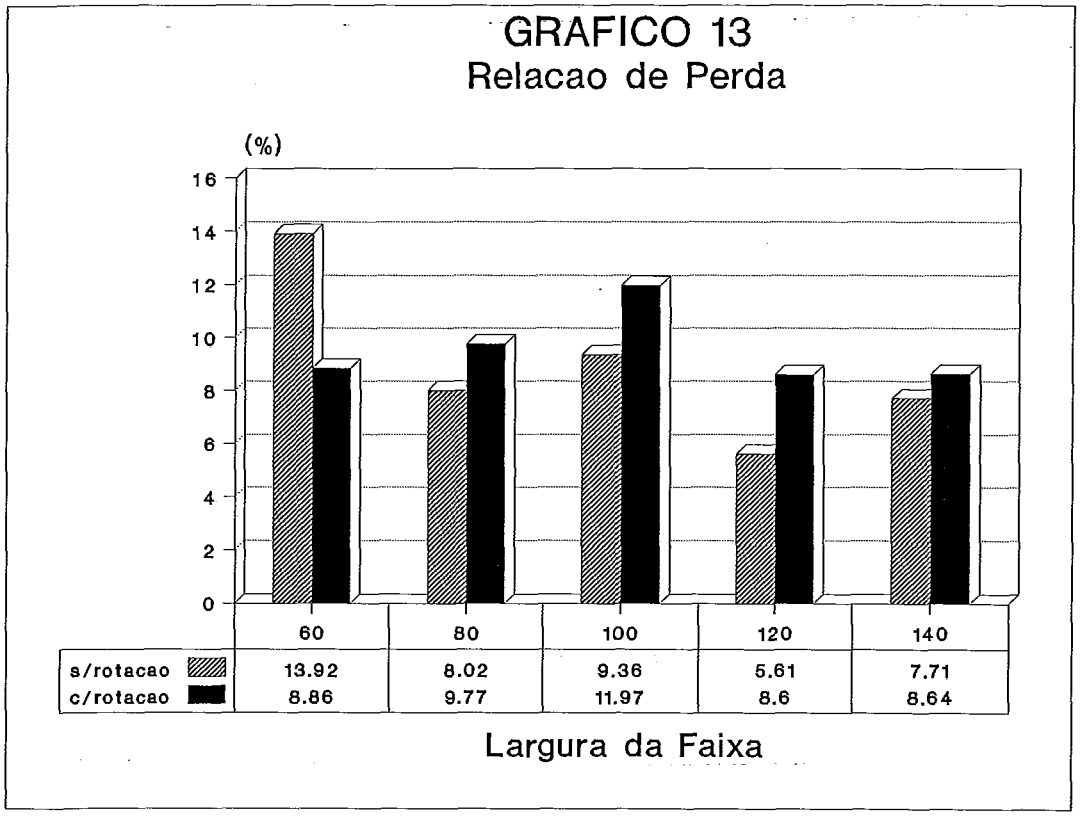


TABELA 15

RESULTADOS CORRESPONDENTES A LISTA DE ENTRADA								
	BI-MMD (s/ ROTACAO)				BI-MMD (c/ ROTACAO)			
FAIXA	AREAC	REND	PERDA	DIST	AREAC	REND	PERDA	DIST
60	88560	86,31%	13,92%	1476	83640	91,14%	8,86%	1394
80	82880	91,98%	8,02%	1036	84480	90,23%	9,77%	1056
100	84100	90,64%	9,36%	841	86600	88,03%	11,97%	866
120	80760	94,39%	5,61%	673	83400	91,40%	8,60%	695
140	82600	92,29%	7,71%	590	83440	91,36%	8,64%	596
MEDIA	83780	91,08%	8,92%	923	84312	90,43%	9,57%	921

Conforme mostra a Tabela 14, à medida que o número de retângulos vai aumentando o rendimento do algoritmo também vai aumentando (ou taxa de perda diminuindo), o que de certa forma mostra o bom desempenho do mesmo. Estas experiências indicaram também, que o aproveitamento médio sobre a faixa, com o aumento no número de ítens ficaram acima de 85%; e com a variação na altura da faixa, mantendo o número de ítens fixado em 100, a média de aproveitamento na faixa continua ainda acima deste valor (veja Tabela 15 e Gráfico 13).

As Figuras V.9 (a) e (b), ilustram a execução do algoritmo BI–MMD para as 10 primeiras caixas da lista de entrada, organizada na Tabela 13. Observe a realização da rotação nas caixas 1,4,7–10 e a mudança nos limites obtidos.

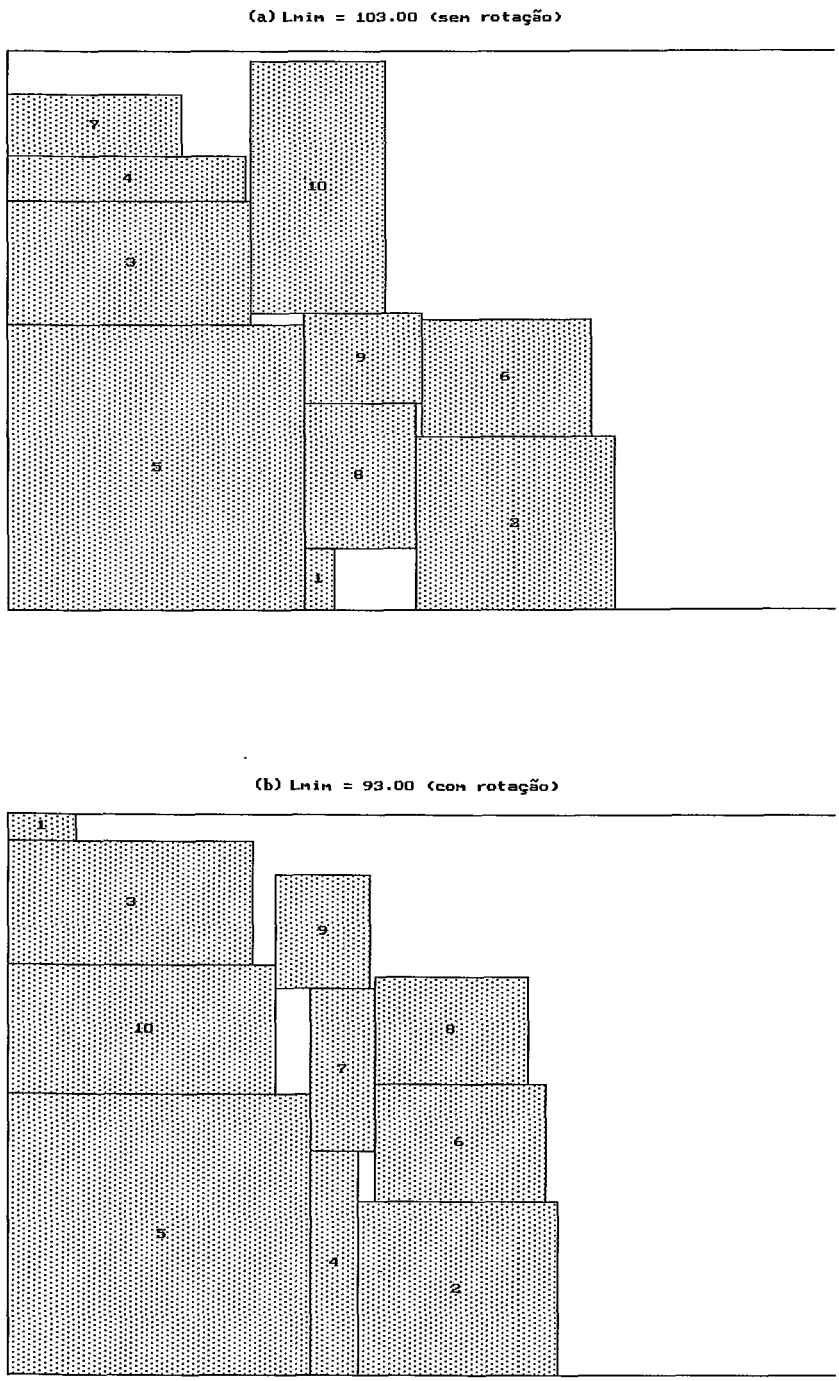


FIG. V.9: *Um exemplo de utilização do algoritmo BI–MMD.*

Para o **natural 2–D Bin–Packing**, nós adaptamos o algoritmo BI–MMD para operar iterativamente, empacotando somente aquelas caixas que satisfaçam ao bin ativo, i.e., que não ultrapassarem a largura horizontal do bin em questão. O algoritmo é aplicado da mesma forma nas caixas restantes para satisfazer um novo bin, e assim por diante. Existe,entretanto duas maneiras de assim se proceder. A primeira delas opera de forma a que um novo bin seja requerido somente quando, um dos empilhamentos que tenha ultrapassado a fronteira horizontal tenha terminado, e os retângulos que não satisfizeram ao bin, são devolvidos a lista para empacotamento de um novo bin (observe Figura V.10(a)); a outra maneira, é quando na formação de uma pilha um retângulo atinge a fronteira horizontal, neste caso, aquele retângulo é devolvido a lista e, então, o bin é encerrado, novo bin é ativado, e assim este processo é repetido, sucessivamente, até que a lista de retângulos tenha se esgotado. (Observe Figura V.10(b)).

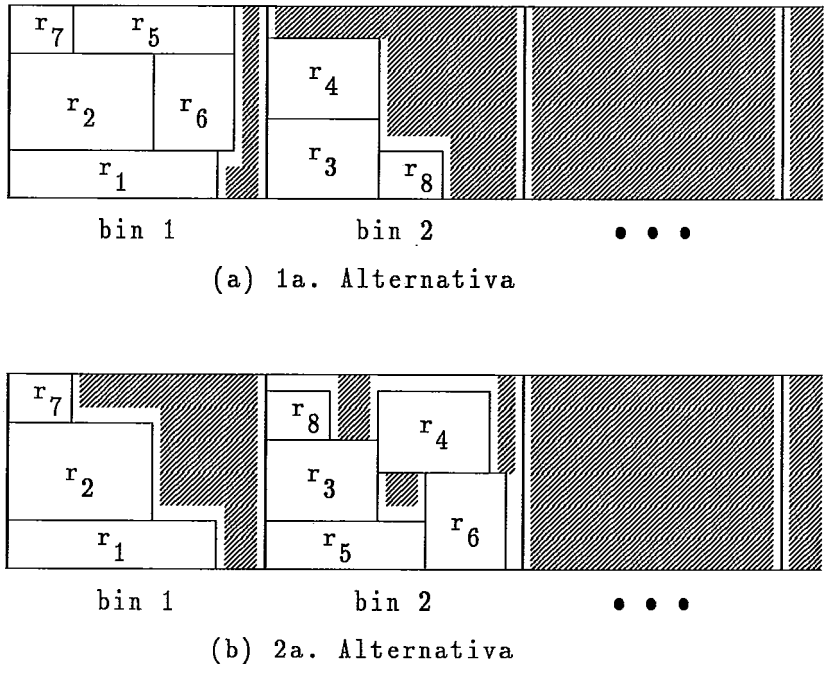


FIG. V.10 (a) e (b): *Duas maneiras de aplicação de BI–MMD em natural 2–D packing.*

Os dados foram gerados aleatoriamente, para testes, através de uma procedure GERADOR 2–D que fraciona os bins (no caso bins de dimensões 100x100 u^2), semelhante àquela da seção 4.1.6, atendendo a mesma finalidade de medida de performance do algoritmo em relação a solução ótima.

Uma variação no ótimo de 10 (ou 5) bins e terminando com 100 (50) bins, dependendo de suas dimensões, foi aplicada nas heurísticas desenvolvidas. A Tabela 16, apresenta uma lista de resultados obtidos para ambas alternativas, **sem** e **com** rotação.

Para estas instâncias, a segunda alternativa de empacotamento mostrou–se melhor que a primeira. Houve uma quase linearidade nas respostas obtidas, e pelo menos para estes dados gerados:

$$\frac{\text{BIMMD}(L)-\text{OPT}(L)}{\text{OPT}(L)} \approx 20\%$$

TABELA 16

RESULTADOS CORRESPONDENTES A LISTA DE ENTRADA GERADA

No. ÓTIMO BINS	BINS (100X100) S/ROT. BIMMD	S/ROT. PEC	C/ROT. BIMMD	C/ROT. PEC	BINS (200X100) S/ROT. BIMMD	S/ROT. PEC	C/ROT. BIMMD	C/ROT. PEC	NÚMERO DE CAIXAS
10 (5)	12	12	12	12	6	6	6	6	50
20 (10)	24	24	24	24	11	**12**	12	**14**	96
30 (15)	36	36	34	34	17	17	17	**18**	142
40 (20)	50	50	46	46	22	22	23	23	202
50 (25)	61	**62**	60	60	28	**29**	28	**29**	280
60 (30)	73	73	67	67	**34**	33	34	34	304
70 (35)	87	87	76	76	38	**40**	39	**49**	352
80 (40)	99	99	88	88	44	**45**	44	44	400
90 (45)	111	111	103	103	50	**51**	52	52	454
100 (50)	124	124	**112**	111	55	55	56	56	488

P.E.C. = Pseudo Empacotamento das Caixas ou 1a alternativa

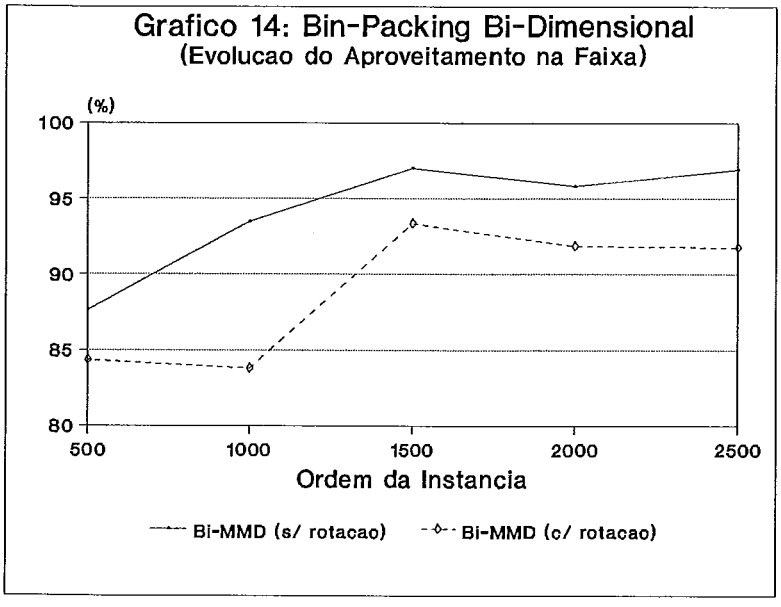
Usando os dados e limites estabelecidos pelos bins definidos na Tabela 16, podemos obter agora a razão de performance do algoritmo aplicado sobre o problema de empacotamento numa faixa. A Tabela 17, expressa os resultados.

TABELA 17

RESULTADOS CORRESPONDENTES A LISTA DE ENTRADA GERADA (%)

LIMITE MÍNIMO FIXADO	FAIXA (ω X 100) BI-MMD SEM ROTAÇÃO REND	PERDA	DIST	RAZÃO	FAIXA (ω X 100) BI-MMD COM ROTAÇÃO REND	PERDA	DIST	RAZÃO	No. DE CAIXAS
500	87,62	12,38	570,65	1,141	84,36	15,64	529,69	1,185	50
1000	93,48	6,52	1069,79	1,070	83,77	16,23	1193,77	1,194	96
1500	97,03	2,97	1545,88	1,031	93,35	6,65	1606,90	1,071	142
2000	95,83	4,17	2086,99	1,043	91,87	8,13	2176,99	1,088	202
2500	96,95	3,05	2578,63	1,031	91,79	8,21	2723,51	1,089	280
MEDIA	94,18	5,82	–	1,063	89,03	10,97	–	1,125	–

Analisando os resultados podemos notar que a performance do algoritmo melhora à medida que o número de caixas cresce, e que o aproveitamento médio ficou acima de 89% para os dois casos. O Gráfico 14 mostra a evolução do aproveitamento na faixa.



5.3 Estudo de um caso Tri–Dimensional

O problema de alocação de caixas em *containers* é um problema tipo *Bin–Packing* tri–dimensional, altamente combinatório e de solução muito complexa.

O que propomos, nesta seção, são algumas idéias para otimizar a alocação do espaço em conteiners ou veículos no transporte de caixas, acondicionadoras de carretéis de fios, na indústria têxtil. Mais especificamente, nos limitamos a uma situação onde a variabilidade nos tipos de caixas e veículos não é muito grande, embora a relação comprimento de caixa versus comprimento de veículo seja significativa acarretando um problema combinatório de grande porte.

O problema em consideração é:

Dados um conjunto de veículos $V = \{v_1(c_1,l_1,h_1,t_1),\ldots,(v_n,l_n,h_n,t_n)\}$ e um conjunto de caixas $C = \{c_1\{x_1,y_1,z_1,p_1),\ldots,c_m(x_m,y_m,z_m,p_m)\}$, otimizar a alocação das caixas $c_j \in C$ em veículos selecionados, onde c_i,l_i,h_i,t_i e x_j,y_j,z_j,p_j são, respectivamente, as medidas de comprimento, largura, altura e capacidade do veículo v_i e da caixa c_j.

A questão da seleção dos veículos mais adequados para transporte e da ordem de prioridade de distribuição aos clientes é uma extensão deste problema e está sendo tratada em estudos fora deste trabalho.

Os procedimentos aqui sugeridos são para geração de faixas e prismas de caixas que servirão como dados de entrada para rotinas de programação linear inteira e mixta, as quais fornecerão faixas prismáticas e ocupação final das mesmas nos veículos selecionados.

5.3.1 O Procedimento Geral

O modelo de aplicação é:

Passo 1: Para cada veículo i selecionado gerar faixas a partir das dimensões das caixas padronizadas e segundo uma eficiência de ocupação $\geq$ uma eficiência pré–estabelecida. Algoritmos aproximativos de bin–packing uni–dimensionais (programa BIN1.EXE) são adaptados para gerar tais faixas. Um banco de arquivos de faixas (ARQ1) é construido para os veículos freqüentemente usados, de acordo com as dimensões dos mesmos.

Passo 2: Inclusão de planos canônicos (se não suprido pelo passo 1) e/ou eliminação de todos os planos correspondentes as caixas com demanda, do dia, nula (programa CANÔNICO.EXE). Criação do arquivo MPLANORE.

Passo 3: A partir do arquivo MPLANORE, um subconjunto de caixas correspondentes a demanda do dia é obtido como lista de entrada para aplicação, novamente, de algoritmos de bin–packing a uma dimensão. Neste caso, os algoritmos aproximativos são adaptados para fazerem uma partição nesta lista, de tal forma a compor pilhas de caixas (prismas), onde a base das menores caixas estejam contidas inteiramente nas bases das caixas maiores. Os bins têm alturas correspondentes as alturas dos veículos. A saída é o arquivo NEWTAB constituído só das caixas que possuem demandas não nulas e com formato apropriado para os dados, ou seja, com a identificação de cada prisma e caixas que o compoem, assim como o volume ocupado.

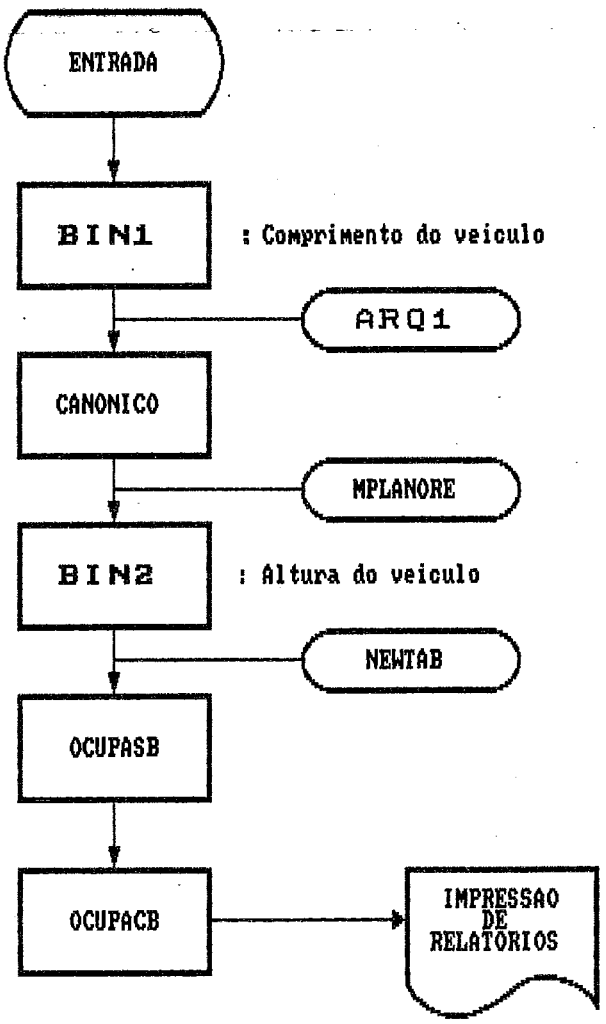
Passo 4: Utilizando–se do arquivo NEWTAB é formulado um problema de programação linear mista (programa OCUPASB.EXE) cuja solução ótima define parâmetros que serão utilizados no modelo definido no passo 5 (i.e.,

estabelece o número de faixas necessárias para melhor ocupação do lastro do veículo). Nesse modelo as variáveis inteiras são as faixas.

Passo 5: Constitui uma simplificação do modelo OCUPASB, contemplando somente as variáveis inteiras, i.e., só considera do modelo anterior as faixas. Introduz um *bound* para cada variável (caixa) restringindo o número de caixas por faixa (programa OCUPACB.EXE). A solução deste novo modelo define a alocação ótima de caixas para o veículo selecionado.

5.3.2 Fluxograma de execução do sistema

O fluxograma abaixo mostra a conexão dos programas:



5.3.3 Descrição da aplicação de Bin–Packing uni–dimensional

Como, normalmente, o número de caixas padrões usado nessas indústrias não é grande, os algoritmos de Bin–Packing uni–dimensionais expostos neste trabalho, podem auxiliar na determinação dos diversos planos, tanto das faixas como das pilhas, para composição do lastro do veículo (FiguraV.11) Isto é vantajoso porque esta determinação é feita de maneira muito rápida, em tempo polinomial. Ao invés de métodos enumerativos, que são exatos porém de tempo de execução muito demorado, muitas vezes levando a empresa a utilizar métodos empíricos.

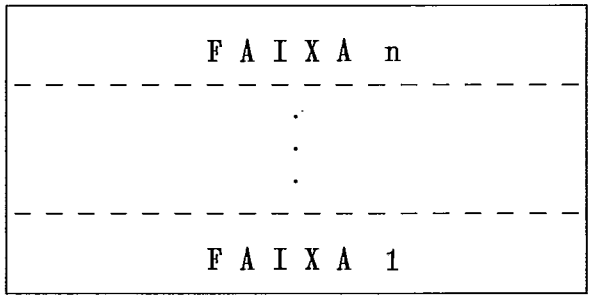


FIG. V.11: *Divisão do lastro em faixas*

Consideramos a seguinte idéia: todas as caixas que irão compor uma faixa (bin) têm inicialmente a mesma espessura que aquela caixa maior que determinou a largura da faixa, o que fica inalterado são apenas os seus tamanhos. Então, o processo é imaginar que para cada faixa existe um número ilimitado de idénticas faixas ou bins. Aplica–se, então, os algoritmos aproximativos, os quais são adaptados para selecionar aqueles empacotamentos nos quais satisfazem a eficiência mínima estabelecidade de ocupação na faixa. Este aproveitamento pode ser determinado pela relação:

$$\frac{\text{Área consumida pelas caixas}}{\text{Área real ocupada pelas caixas}} \times \% \geq \text{eficiência estabelecida}$$

O resultado é, portanto, sequencial por faixa, descartando sempre as combinações de baixa eficiência.

O mesmo raciocínio pode ser aplicado na formação das pilhas.

Ressaltamos, que todos os algoritmos aproximativos uni–dimensionais, abordados neste trabalho, têm idêntica importância aqui uma vez que podem gerar diferentes combinações ou planos de caixas para composição das faixas ou prismas.

Capítulo VI
Conclusões e Sugestões

Apresentamos, neste trabalho, um estudo detalhado sobre o problema Bin–Packing. Mostramos a evolução das pesquisas deste problema tanto pelo lado de métodos exatos de resolução quanto da abordagem de algoritmos aproximativos, cujas soluções são sub–ótimas (ou, bem próximas do ótimo). Nos dois primeiro modelos exatos, desenvolvemos a parte de relaxação e decomposição Lagrangeanas, visando um procedimento Branch–and–Bound, por não termos conhecimento de início, destas técnicas aplicadas no Bin Packing Generalizado de Lewis & Parker [54], o qual exploraram extensivamente esse assunto. Excluindo a Decomposição Lagrangeana.

Procuramos estabelecer a semelhança ou diferença do problema de Bin–Packing com outros problema de Cutting & Packing. Várias variantes deste problema foram discutidas e suas formulações matemáticas apresentadas.

O problema Bin Packing, como um problema de decisão, está na classe dos problemas NP–completos, o que permitiu a nossa concentração pela procura por novos algoritmos aproximativos de tempo polinomial. Como fruto disto, desenvolvemos uma classe de algoritmos aproximativos decrescente, com base em um algoritmo chave denominado Maiores e Menores Decrescente (MMD). A performance e complexidades computacionais, como é de praxe, na validação de algoritmos aproximativos foram analisadas, bem como testes empíricos usando distribuições uniformes obtidos através de geração aleatória de dados e, também, pela construção de listas construídas a partir da solução ótima conhecida à priori. Vários testes comprovam a eficiência, na prática, dos resultados desta classe de algoritmos *off–line* uni–dimensionais. Em continuação a linha de trabalho que traçamos, um algoritmo Bi–Dimensional, extensão natural do algoritmo Maiores e Menores Decrescente foi desenvolvida, e testes de qualidades de respostas foram realizados, em uma amostra, simplesmente pela evolução da taxa de perda ou de

rendimento de aproveitamento na faixa, quando o número de caixas era elevado passo a passo. O resultado mostrou que a performance assintótica tende a um valor satisfatório. Isto é, a taxa de perda, em média decresce, com o aumento no número de caixas. Como contribuição final, abordamos um método heurístico para otimizar a alocação do espaço em containers em veículos no transporte de fios acondicionados em caixas na indústria têxtil. Esse método faz uso dos algoritmos aproximativos sobre bin packing desenvolvido neste trabalho.

Uma forma de eliminar o inconveniente da ordenação dos ítens em algoritmos *off–line* é particionando–os dentro de grupos ou classes de acordo com os seus valores e elaborar algoritmos que eficientemente combine estas classes formando os empacotamentos. Um algoritmo que assim procede é o *Group–X–Fit Grouped* apresentado por JOHNSON em [42], com uma razão de performance no pior–caso de 1.5. Porém, outros refinamentos de grupos ou classes podem ser pensados para melhorar ainda mais este limite de performance de pior–caso, mantendo a complexidade do algoritmo em $O(n)$. Os esquemas de aproximação de Fernandez de La Vega & Lueker [23] e Kamarkar & Karp [46], mostraram que para qualquer $\epsilon > 0$ existe um algoritmo A tal que $A(\mathrm{L}) \leq (1 + \epsilon)L^* + C_\epsilon$, e A rodando em tempo de $O(n) + D_\epsilon$, onde C_ϵ e D_ϵ são funções que dependem de ϵ. No primeiro caso a função tem crescimento exponencial, e no segundo, crescimento polinomial em ϵ, mas o polinômio é de ordem muito grande, o que torna inviável o uso destes esquemas na prática.

Há em andamento um estudo de aproveitamento do algoritmo Bi–MMD em rotinas de saídas gráficas, onde ao terminar o processo de execução do algoritmo o resultado é exposto de forma gráfica, mostrando os empacotamentos e permitindo também que seja possível, com o uso de **mouse** ou **canetas óticas**, efetuar mudanças nessa resposta final para melhorar ainda mais as disposições dos cortes ou empacotamentos. O estudo do aproveitamento destes algoritmos se encontra ainda em fase preliminar. Testes estão sendo realizados com a idéia de combinar os ganhos obtidos dos algoritmos **sem** e **com** ordenação para obtermos um único algoritmo que melhor

aproveite as extensões da faixa ou bin em cada iteração.

Como sugestão aconselhamos a implementação do método dual ascendente Lagrangeano, como apresentado em GUIGNARD & ROSENWEIN [34], para obtenção de um bound melhorado na aplicação de um dos métodos de branch–and–bound já desenvolvidos e comentados neste trabalho. A implementação usando o método de subgradiente normal não tem até agora justificado a sua utilização, pois exige um número muito grande de iterações duais. Um estudo mais completo sobre a análise probabilística, área mais recente da investigação da performance dos algoritmos aproximativos do problema bin–packing, está pouco explorada e um survey completo sobre a mesma não foi ainda publicado. Além disso, a análise probabilística do algoritmo MMD ficou em aberto para futuras pesquisas.

Como os algoritmos, na literatura especializada sobre bin–packing, trabalham empacotando os ítens na sequência em que são dados na lista de entrada, acreditamos que a nossa iniciativa de mudar esta regra para algoritmos *off–line*, trabalhando com elementos intermediários, fugindo ao sequenciamento, abre um caminho para outros algoritmos aproximativos para a resolução do problema.

A utilização de bin–packing a problemas de roteamento de veículos merece uma atenção especial, uma vez que estes problemas estão relacionados em diversas situações práticas (p. ex., no caso 3–D de escolha de carretas para o transporte de caixas padronizadas e entrega em diversas localidades de demanda) e o desenvolvimento de heurísticas envolvendo este casamento poderá trazer bons resultados, o qual ambos poderiam não corresponder ao desejado quando resolvidos separadamente.

O desenvolvimento de algoritmos aproximativos duais bi–dimensionais são ainda desconhecidos. O uso de Teoria dos Grafos em bin–packing tem sido muito restrito. Existe, portanto, uma série de situações a serem exploradas.

Durante o nosso estudo pudemos perceber que a partir da década de 80 grande parte das pesquisas voltaram–se para a análise probabilística. À princípio, o interesse voltou–se para as heurísticas clássicas anteriormente analisadas pelo pior–caso. Por

exemplo, FREDERICKSON [26] e LUEKER [55] analisaram a performance das heurísticas First–Fit Decreasing e Best–Fit Decreasing sob a distribuição uniforme U(0,1]. Eles mostraram que ambos algoritmos são assintóticamente ótimos, no sentido que a razão do número esperado de bins empacotados por qualquer uma destas heurísticas para o número exato de bins que é n/2, converge para 1. COFFMAN *et al.* [17] analisaram a performance esperada do algoritmo Next Fit para um número infinito de elementos independentes cujos tamanhos eram uniformemente distribuidos, e concluiram que a performance esperada da heurística é limitada por 4/3 do ótimo esperado. Paralelamente, estudos para obtenção de algoritmos aproximativos de bin–packing bi–dimensional tiveram início, com os trabalhos já citados no Cap. V, e a perquisa para a análise da performance do caso médio dos algoritmos heurísticos bi–dimensionais tiveram grande impulso. A análise do pior–caso, de grande parte destes algoritmos estão em aberto. Atualmente, a busca se concentra em encontrar algoritmos bi–dimensionais cada vez melhores com relação a análise do caso–médio. No entanto, continuam ainda surgindo tanto algoritmos uni–dimensionais, análise do pior–caso dos mesmos, como também, resultados para algoritmos exatos. Neste último caso é possível citar o algoritmo exato, para o caso uni–dimensional, desenvolvido por MARTELLO & TOTH [58].

BIBLIOGRAFIA

[1] ASSMANN, S. B., JOHNSON, D. S., KLEITMAN, D. J. and LEUNG, J. Y. T., "On a dual version of the one–dimensional bin packing problem", *J. of Algorithms 5* (1984), 502 – 525.

[2] BAKER, B. S. and COFFMAN, JR, E. G. – "A Tight Asymptotic Bound on Next–Fit–Decreasing Bin–Packing", *SIAM J. on Algebraic and Discrete Methods 2* (1981), 147–152.

[3] BAKER, B. S., BROWN, D. J., and KATSEFF, H. P., "A 5/4 algorithm for two–dimensional Packing," *J. Algorithms, 2* (1981), 348–368.

[4] BAKER, B. S., COFFMAN, JR. E. G. and RIVEST, R. L., "Orthogonal Packing in Two Dimensions," *SIAM J. Comput 9* (1980), 845–855.

[5] BALAS, E., "An Additive Algorithm for Solving Linear Programs with Zero–One variables," *Operations Research, 13* (1965), 517–546.

[6] BROWN, A. R., *Optimun Packing and Deplecion*, American Elsevier, New York (1971).

[7] BROWN, D. J., "An Improved BL Lower Bound," *Inf. Proc. Letters 11* (1980), 37–39.

[8] CHUNG, F. R. K., GAREY, M. R., and JOHNSON, D. S., "On packing two–dimensional bins," *SIAM J. Alg. Disc. Meth. 3* (1982), 66–76.

[9] CHVATAL, V., *Linear Programming*, W. H. Freeman, New York (1983).

[10] CHVATAL, V., and ARMSTRONG, W., "Programmation Heuristique" *Notes de cours*, Université de Montréal, 1980.

[11] COFFMAN JR. E. G. - "An Introduction to proof Techniques for Bin–Packing Approximation Algoritms", in *Deterministic and Stochastic Scheduling*, (M. A. H. Dempster et al. eds, Reidel Publishing Company, 1982).

[12] COFFMAN JR.R. E. G. and SHOR, P. W. - "Average–case analysis of cutting and packing in two dimensions", *Eur. J. Oper. Res. 44* (1990), 134–144.

[13] COFFMAN, E. G., Jr., GAREY, M. R., and JOHNSON, D. S., "Dynamic bin packing," *SIAM J. Comput. 12* (1983), 227–258.

[14] COFFMAN JR., E. G., GAREY, M. R. and JOHNSON, D. S. – "Approximation Algorithms for Bin Packing An Updated Survey", in *Algorithm Design for Computer System Design.* (Ausiello, G., Lucertini, M. and Serafini, P., Eds.), Springer–Verlag, New York, (1984).

[15] COFFMAN JR., E. G., GAREY, M. R. and JOHNSON, D. S. – "Bin Packing with Divisible Item Sizes", *J. of Complexity 3* (1987), 406–428.

[16] COFFMAN JR., E. G., GAREY, M. R., JOHNSON, D. S. and TARJAN, "Performance Bounds for Level–Oriented Two–Dimensional Packing Algorithms," *SIAM J. Comput. 9* (1980), 808–826

[17] COFFMAN, E. G., HOFRI, M. and YAO, A. C. – "A Stochastic Model of Bin–Packing," *Inform. Control 44* (1980), 105–115.

[18] COFFMAN JR., E. G. and LAGARIAS, J. C.,"Algorithms for Packing Squares: A Probabilistic Analysis," *SIAM J. Comput. 18* (1989), 166–185.

[19] COFFMAN, E. G., Jr., LEUNG, J. Y., and TING, D. W., "Bin packing: maximizing the number of pieces packed," *Acta Informatica 9* (1978), 263–271.

[20] COOK, S. A., "An Overview of Computacional Complexity," *Communications of the ACM 26(6)*, Jun. 1983.

[21] DYCKHOFF, H., "A Typology of Cutting and Packing problems", *European Journal of Operations Research 44* (1990), 145–159.

[22] EILON, S. and CHRISTOFIDES, N., "The loading problem," *Management Sci. 17* (1971), 259–268.

[23] FERNANDES De La VEGA, W. and LUEKER, G. S., "Bin packing can be solved within $1+\epsilon$ in linear time," *Combinatorica 1* (1981).

[24] FISHER, M. L., "Worst–Case Analysis of Heuristic Algorithms," *Management Sci. 1* (1980), 1–17.

[25] FISHER,M. L., NORTHUP, W. and SHAPIRO, J., "Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience," *Mathematical Programming Study 3*, North Holland Publishing Co., Amsterdam, (1975), 56–94.

[26] FREDERICKSON, G. N., "Probabilistic Analysis for Simple One and–Two–Dimensional Bin Packing Algorithms", *Inf. Proc. Letters 11* (1980), 156–161.

[27] FRIESEN, D. K. and LANGSTON, M. A., "Variable sized bin packing," *SIAM J. Oomput. 15* (1986) 222–230.

[28] GAREY, M. R. and JOHNSON, D. S., *Computers and Intractability: A Guide to the Theory of NP–Completeness*, W. H. Freeman and San Francisco (1979).

[29] GAREY, M. R., GRAHAM, R. L. and JOHNSON, D. S., "On a number–theoric bin packing conjecture," *Proc. 5th Hungarian Combinatorics Colloquium*, North–Holland, Amsterdam (1978), *377*–392.

[30] GAREY, M. R., GRAHAM, R. L., JOHNSON, D. S. and YAO, A. C., "Resource Constrained Scheduling as Generalized Bin Packing," *J. Combinatorial Theory Ser. A 21* (1976), 257–298.

[31] GEOFFRION, A., "Lagrangian Relaxation for Integer Programming," in *Mathematical Programming Study 2, Approaches to Integer Programming,* M. L. Balinski, Ed. 82–114 (North–Holland Publishing Co., Amstterdam 1974).

[32] GRAHAM, R. L., "Bounds on multiprocessing anomalies and related packing problem" in: *Procceedings AFIPS Spring Joint Computer Conference* (1972), 205–217.

[33] GROTSCHEL, M. LOVASZ, L. and SCHRIJVER A., "The ellipsoid method and its consequences in combinatorial optmization", *Combinatorica 1* (1981), 169–197.

[34] GUIGNARD M. and ROSENWEIN, M. B., "An application–oriented guide for designing Lagrangean dual ascendent algorithms", *European J. Oper. Res. 43* (1989), 197–203.

[35] HELD, M., WOLFE, P. and CROWDER, H., "Validation of Subgradiente Optimization", *Mathematical Programming 6* (1974), 62–88.

[36] HOROWITZ, E. & SAHNI, S. *Fundamentals of Data Structures in Pascal*, 3d ed., Computer Science Press (1990).

[37] HU, T. C., *Combinatorial Algorithms*, Addison–Wesley, (1982).

[38] HUNG, M. S. and BROWN, J. R., "An Algorithm for a class of loading problems", *Naval Res. Logist. Quart. 25* (1978), 289–297.

[39] HUNG, M. S. and FISK, J., "An Algorithm for 0–1 Multiple Knapsack Problems", *Naval Res. Logist. Quart. 25* (1978), 571–579.

[40] INGARGIOLA, G. and KORSH, "An Algoritm for the Solution of 0–1 Loading Problems," *Operations Reseach, 23* (1975), 1110–1119.

[41] JOHNSON, D. S., "Approximation Algorithms for Combinatorial Problems," *J. of Computer and Sciences, 9* (1974), 256–278.

[42] JOHNSON, D. S., "Fast Algorithms for Bin Packing," *J. Comput. Syst. Sci.* 8 (1974), 272–314.

[43] JOHNSON, D. S., "Near optimal bin packing algorithm." Ph.D. Th., M.I.T., Cambridge, Mass., June 1973.

[44] JOHNSON, D. S., DEMERS, A., ULLMAN, J. D., GAREY, M. R. and GRAHAM, R. L., "Worst–case performance bounds for simple one–dimensional packing algorithms," *SIAM J. Comput. 3* (1974), 299–325.

[45] JOHNSON, D. S. and GAREY, M. R., "A 71/60 Theorem for Bin Packing." *Journal of Complexity 1* (1985), 65–106

[46] KAMARKAR, N. and KARP, R. M., "An efficient Approximation scheme for the one–dimensional bin packing problem," *Proc. 23 rd Ann. Symp. on Foundations os Comp. Science,* IEEE Computer Soc., Nov. (1982), 312 – 320.

[47] KARP, R. M., "Reducibility among combinatorial problems," in R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Presss, New York (1972), 85–103.

[48] KHACHIYAN, L. G., "A polynomial algorithm in linear programming," *in Soviet Math. Dokl. 20* (1979), *191–194.*

[49] KINNERSLEY, N. G. and LANGSTON, M. A., "Online Variable–Sized Bin–Packing" *Discrete Applied Mathematics 22* (1988/89) 143–148.

[50] KRAUSE, K. L., SHEN, Y. Y., and SCHEWETMAN, H. D., "Analisys of several task scheduling algorithms for a model of multiprogramming computer systems," *J. Assoc. Comput. Mach. 22* (1975), 522–550.

[51] LANGSTON, M. A., "Performance of Bin Packing Heuristics for Maximizing the number of Pieces Packed into Bins of Different Sizes," Working paper No. 90 (1982), Dept. Comput. Sci., Washington State University, Washington.

[52] LASDON, L. S., *Optimization Theory for Large Systems*, Macmillan, Ney York, (1970), 523p.

[53] LEWIS, R. T., "A Generalized Bin–Packing Problem," Doctoral Thesis, Gorgia nstitute of Technology, Atlanta (1980).

[54] LEWIS, R. T. and PARKER, R. G., "On a Generalized Bin–Packing Problem", *Nav. Res. Log. Quarterly 29* (1982), 119–145.

[55] LUEKER, B. S., "Bin Packing with Items Uniformly Distributed over Intervals [a,b]", *Proc. 24th Annual Sympos. Foundations of Computer Science*, Tucson, AZ, (1983), 289–297.

[56] MACULAN, N., CAMPELLO, R. E. e RODRIGUES, L. B. – "Relaxação Lagrangeana em Programação Inteira", *Relatorio Técnico ES40–84,* COPPE/UFRJ.

[57] MAGAZINE, M. J. and WEE, T. S., "The generalization of bin–packing heuristics to the line balancing problem," Working Paper No. 128 (1979), Dept. Mgmt. Sci., University of woterloo, Woterloo, Ontario.

[58] MARTELLO, S. and TOTH, P., *Knapsacks Problems*, John Wiley & Sons (1990)

[59] MARTELLO, S. and TOTH, P., "Solution of the zero–one multiple knapsack problem" *Eur. J. Oper. Res. 4* (1980), 276–283.

[60] MARTELLO, S. and TOTH, P., "A Program for the 0–1 Multiple Knapsack Problem," *ACM Trans. Math. Systems, 11.2* (1985), 135–140.

[61] MARTELLO, S. and TOTH, P., "Lower bounds and Reduction Procedures for the Bin–Packing Problem," *Discret Applied Mathematics 28* (1990), 59–70.

[62] MINOUX, M., *Mathematical Programming*, John Wiley and Sons Ltd,. (1986).

[63] Murgolo F. D., "An Eficient Approximation Scheme for Variable–Sized Bin–Packing, *SIAM J. Comput. 16* (1987) 149–161.

[64] NYHOFF, L. and LEESTMA, S., *Advanced Programming in Pascal with Data Structures*, Macmillan, Inc., (1988).

[65] SHANI, S. and GONZALES, T., "P–complete aproximation problems", *J. ACM 23* (1976), 555–565.

[66] SHEARER, J. B., "A Counterexample to Bin Packing Conjecture", *SIAM J. Alg. Disc. Meth. 2* (1981), 309–310.

[67] SLEATOR, D.K.D.B, "A 2.5 times optimal algorithms for bin packing in two dimensions," *Information Processing Lett. 10* (1980), 37–40.

[68] SYSLO, M. M., *Discrete optimization algorithms*, Prentice–Hall, Inc., Englewood Cliffs, New Jersey (1983).

[69] TOSCANI, L. V., e SZWARCFITER, J. L., "Algoritmos Aproximativos," Relatório de Pesquisa No. 54 (1986), NCE–UFRJ, Rio de Janeiro.

[70] YAO, A. C., "New algorithms for bin packing", *J. Assoc. Comput. Mach. 27* (1980), 207–227.

APÊNDICE A

Demonstração das propriedades da função w(x) apresentadas na prova do teorema 2.

A demonstração daquelas propriedades é um pouco trabalhosa e se baseia em algumas proposições (ver CHVATAL [CH]):

Proposição 1: Se $\frac{1}{3} \leq x \leq \frac{1}{2}$ então $w(\frac{1}{3}) + w(x - \frac{1}{3}) = w(x)$.
Prova: Desde que a inclinação de w(x) é a mesma nas regiões $(0,\frac{1}{6}]$ e $[\frac{1}{3},\frac{1}{2}]$, podemos substituir x pela soma de dois números $\frac{1}{3}$ e $x - \frac{1}{3}$, de forma que o peso em x é a soma dos pesos naqueles números.■

Proposição 2: Se $0 \leq x, y \leq \frac{1}{6}$ então $w(x) + w(y) \leq w(x+y)$.
Prova: É imediata, pela própria construção da função.■

Proposição 3: Se $0 \leq x \leq \frac{1}{2}$ então $\frac{6}{5}x \leq w(x) \leq \frac{3}{2}x$.
Prova: No intervalo $(0,\frac{1}{6}]$, $\frac{w(x)}{x} \geq \frac{6}{5}$. Nos intervalos $(\frac{1}{6},\frac{1}{3}]$ e $(\frac{1}{3},\frac{1}{2}]$, os valores extremos de $\frac{w(x)}{x}$ são atingidos em $x = \frac{1}{3}$ (valendo $\frac{3}{2}$) e $x = \frac{1}{2}$ (valendo $\frac{7}{5}$), respectivamente. Como são maiores do que $\frac{6}{5}$, a prova está concluída.■

Proposição 4: Se $x \leq x^* \leq \frac{1}{2}$, então $w(x) \leq \frac{9}{5}(x^* - x)$.
Prova: O resultado é óbvio, desde que a inclinação dos segmentos de retas em cada um dos três subintervalos de $0 \leq x \leq \frac{1}{2}$ não excede $\frac{9}{5}$.■

Proposição 5: Se $k \geq 2$, $x_1 \geq x_2 \geq \ldots \geq x_k$ e $1 - x_k \leq x_1 + x_2 + \ldots + x_k \leq 1$, entdo $w(x_1) + \ldots + w(x_k) \geq 1$.
Prova: Merece um pouco mais de atenção. Se precisamos trabalhar com

$$x_1 \leq \frac{1}{2},\ x_2 < \frac{1}{3} \text{ e que } x_k > \frac{1}{6}.$$

Com efeito, se $x_1 > \frac{1}{2}$ então o resultado é imediato desde que $w(x_1) = 1$. Portanto, admitiremos que $x_1 \leq \frac{1}{2}$. Se $x_2 \leq \frac{1}{3}$ (com $x_1 \geq x_2$), então $w(x_1) + w(x_2) \geq 2w(\frac{1}{3}) =$

Se $x_1 \leq \frac{1}{2}$, $x_2 < \frac{1}{3}$ mas $x_k < \frac{1}{6}$, então $x_1 + ... + x_k \geq 1 - x_k \geq \frac{5}{6}$ e de acordo com a proposição 3, $w(x_1)+...+w(x_k) \geq \frac{6}{5}(x_1+...+x_k) \geq 1$.

Para $k = 2$, temos que $x_1 + x_2 \geq 1 - x_2$ implica em $x_1 \geq \frac{1}{3}$ e $x_2 \geq (1 - x_1)/2$. Caso contrário, invibializaria $x_1 \geq x_2$. Logo,

$$w(x_1) + w(x_2) = \frac{6}{5}x_2 + \frac{1}{10} + \frac{9}{5}x_2 - \frac{1}{10} \geq \frac{6}{5}x_2 + \frac{9}{10}(1 - x_1) \geq 1.$$

Consideremos agora o caso $k \geq 3$. Se $x_1 \geq \frac{1}{3}$ então desde que $x_k > \frac{1}{6}$ e $x_2 < \frac{1}{3}$ (por hipótese)

$$w(x_1) + ... + w(x_k) \geq \frac{6}{5}x_2 + \frac{1}{10} + \frac{9}{5}x_2 - \frac{1}{10} + \frac{6}{5}(x_3+...+x_k) =$$
$$= \frac{6}{5}(x_1+...+x_k) + \frac{3}{2}x_2 \geq \frac{6}{5}(1 - x_k) + \frac{3}{5}x_k \geq 1;$$

enquanto que se $x_1 < \frac{1}{3}$ então

$$w(x_1) + ... + w(x_k) \geq \frac{9}{5}(x_1 + x_2) - \frac{1}{5} + \frac{6}{5}(x_3 + ... + x_k) =$$
$$= \frac{6}{5}(x_1 + ... + x_k) + \frac{3}{5}(x_1 + x_2) - \frac{1}{5} \geq \frac{6}{5}(1 - x_k) + \frac{6}{5}x_k - \frac{1}{5} = 1.\blacksquare$$

Proposição 6: Se $x_1 + x_2 + ... + x_k \leq \frac{1}{2}$, então

$$w(x_1) + ... + w(x_k) \leq \frac{7}{10}.$$

Prova: Podemos supor que $x_i \leq \frac{1}{3}$ para todo i, e de acordo com a proposição 1, cada x_i pode ser substituído pelo par $x_i' = \frac{1}{3}$ e $x_i'' = x_i - \frac{1}{3}$. Podemos supor, pela proposição 2, que ao menos um $x_i \leq \frac{1}{6}$, pois $x_i, x_j \leq \frac{1}{6}$ pode ser reagrupado em $x_i + x_j$.

Quatro casos agora podem ser distingüidos:

(i) $k = 1$ e $x_1 \leq \frac{1}{3}$;

(ii) $k = 2$ e $\frac{1}{6} \leq x_2 \leq x_1 \leq \frac{1}{3}$;

(iii) $k = 2$ e $x_2 \leq \frac{1}{6} \leq x_1 \leq \frac{1}{3}$;

(iv) $k = 3$ e $x_3 \leq \frac{1}{6} \leq x_2 \leq x_1 \leq \frac{1}{3}$.

No caso (i), $w(x_1) \leq w(\frac{1}{3}) = \frac{1}{2} < \frac{7}{10}$.

No caso (ii), temos $w(x_1) + w(x_2) = \frac{9}{5}(x_1 + x_2) - \frac{2}{10} \leq \frac{7}{10}$.

No caso (iii), temos $w(x_1) + w(x_{2)} \leq w(\frac{1}{3}) + w(\frac{1}{6}) = \frac{7}{10}$.

No caso (iv), temos

$$w(x_1) + w(x_2) + w(x_3) = \frac{9}{5}(x_1 + x_2) - \frac{2}{10} + \frac{6}{5}x_3$$
$$\leq \frac{9}{5}(x_1 + x_2 + x_3) - \frac{2}{10} \leq \frac{7}{10}. \blacksquare$$

Agora estamos prontos para a demonstração das propriedades.

Propriedade 1. Se $1/2 < x \leq 1$, então $w(x) = 1$.

Dem: Por definição da função $w(x)$.

Propriedade 2. Se $k \geq 2$, $x_1 \geq x_2 \geq \ldots \geq x_k$, $x_1 + x_2 + \ldots + x_k \leq 1$ e $w(x_1) + \ldots + w(x_k) = 1 - s$ para $s > 0$, então

$$x_1 + x_2 + \ldots + x_k \leq 1 - x_k - \frac{5}{9}s.$$

Dem: Seja t definido por $\Sigma_{i=1}^{k} x_i = 1 - x_k - t$. De acordo com proposição 5, temos $t > 0$. Uma vez que $x_1 + x_2 + t \leq 1 - x_k < 1$, existem x_1^* e x_2^* tais que $x_1 \leq x_1^* \leq \frac{1}{2}$, $x_2 \leq x_2^* \leq \frac{1}{2}$, $x_1^* + x_2^* = x_1 + x_2 + t$.

Tomando $x_i^* = x_i$ para $i = 3, 4, \ldots, k$ obtemos pela proposição 5

$$w(x_1^*) + w(x_2^*) + \ldots + w(x_k^*) \geq 1$$

logo

$$w(x_1^*) + w(x_2^*) \geq w(x_1^*) + w(x_2^*) + s.$$

Já que a proposição 4 implica em

$$s \leq w(x_1^*) - w(x_1) + w(x_2^*) - w(x_2) \leq \frac{9}{5}(x_1^* - x_1) + \frac{9}{5}(x_2^* - x_2) = \frac{9}{5}t,$$

segue que $t \geq \frac{5}{9}s$. Substituindo esta desigualdade na definição de t, a prova fica completa. $\blacksquare$

Proposição 3. Se $x_1 + x_2 + \ldots + x_k \leq 1$, então

$$w(x_1) + w(x_2) + \ldots + w(x_k) \leq \frac{17}{10}.$$

Dem: Se $x_1 \leq \frac{1}{2}$ para todo i, então a proposição 3 implica em

$$\Sigma_{i=1}^{k} w(x_i) \leq \frac{3}{2} \Sigma_{i=1}^{k} x_i \leq \frac{3}{2} < \frac{17}{10} .$$

Se por outro lado, um dos $x_i > \frac{1}{2}$ (digamos x_1), então a proposição 7 se extende a

$$w(x_1) + w(x_2) + \ldots + w(x_k) \leq \frac{17}{10} \cdot \blacksquare$$