Diferentes abordagens para problemas de empacotamento

MAC499 – Trabalho de Formatura Supervisionado

Universidade de São Paulo

Instituto de Matemática e Estatística

Departamento de Ciência da Computação

Dezembro de 2006

Rafael Durbano Lobato[1]

Ernesto G. Birgin (orientador)

[1]Apoio financeiro: FAPESP (processo número 05/57984-6)

Resumo

O problema clássico de empacotamento de retângulos em retângulos consiste em arranjar ortogonalmente itens retangulares, de dimensões (l,w), num retângulo maior, de dimensões (L,W), sem sobreposição. O objetivo é determinar uma disposição com a maior quantidade possível de itens. Conjectura-se que a versão não-guilhotinada do problema é NP-difícil. Neste trabalho abordamos esse problema e estudamos dois algoritmos heurísticos para resolvê-lo.

Palavras-chave: Empacotamento, programação dinâmica, algoritmos recursivos.

Sumário

Parte I

Parte Técnica

Capítulo 1

Introdução

O problema de carregamento de paletes (PCP) consiste em carregar caixas retangulares sobre um palete retangular. Supõe-se que as caixas, disponíveis em grandes quantidades, devem ser arranjadas ortogonalmente, isto é, com um de seus lados ortogonal a um dos lados do palete, e em camadas nas quais a orientação vertical das caixas é fixada. O PCP tem muitas aplicações práticas [10] e aparece com freqüência na logística de armazenamento e transporte de produtos. Um pequeno aumento no número de caixas transportadas pode representar uma redução significativa de custos.

Neste estudo abordamos o problema bidimensional de carregamento de paletes do produtor [11]. Esse problema consiste em arranjar, sem sobreposição, caixas retangulares idênticas num palete retangular. As caixas devem ser colocadas ortogonalmente e podem sofrer rotações de 90°. O objetivo é determinar um arranjo com a maior quantidade possível de caixas. Denotaremos por L e W o comprimento e a largura do palete e por l e w o comprimento e a largura das caixas, respectivamente. Consideramos L, W, l, w inteiros e $L \geq W$ e $l \geq w$, sem perda de generalidade. Assim, cada problema fica determinado pela quádrupla (L, W, l, w). Também podemos supor, sem perda de generalidade, que o máximo divisor comum d de L, W, l e w é 1. Se d for diferente de 1, dividimos as dimensões do palete e das caixas por d e obtemos um problema equivalente.

De acordo com Dyckhoff [10], esse problema pode ser classificado como 2/B/O/C (bidimensional, seleção de itens (neste caso as caixas), apenas um grande objeto (neste caso o palete) e itens de dimensões idênticas).

Bischoff e Dowsland [6] apresentaram uma heurística, baseada nos métodos de Steudel [17] e de De Cani e Smith [7], onde o palete é dividido em no máximo cinco retângulos, cada um dos quais possuindo uma orientação fixa para o empacotamento das caixas, formando padrões não-guilhotinados de primeira ordem.

Morabito e Morales [14] propuseram uma extensão para essa heurística, tornando-a mais geral. Nela, o palete também é dividido em no máximo cinco retângulos. Porém, o método é aplicado recursivamente em cada um dos retângulos gerados.

Lins, Lins e Morabito [13] apresentaram um nova abordagem para resolver o problema. O palete, além de ser dividido em retângulos, como em [14], também é dividido em peças em forma de L. Em experimentos realizados com conjuntos de instâncias do problema bem conhecidos na literatura, incluindo instâncias não resolvidas por outras heurísticas, o método encontrou a solução ótima de todas elas e os autores conjecturaram que essa abordagem sempre encontra empacotamentos ótimos de retângulos idênticos em peças retangulares.

No Capítulo 3, apresentamos o algoritmo proposto em [14] e o conjunto de *raster points* [16]. Fazemos uma análise das simetrias envolvidas nas divisões do palete, propomos algumas melhorias e exibimos os resultados alcançados. No Capítulo 4, descrevemos a abordagem em L e mostramos alguns resultados obtidos com a introdução de *raster points* na implementação do algoritmo. No Capítulo 5, descrevemos os limitantes inferior e superior para o número de caixas (l,w) que podem ser colocadas em um palete.

Capítulo 2

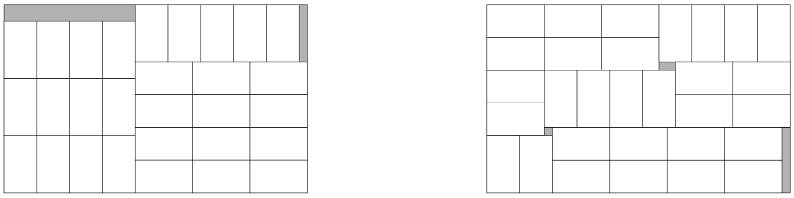
Definições e notações

Neste capítulo apresentamos algumas definições e notações que serão utilizadas daqui em diante.

2.1 Definições

Dizemos que um corte é do tipo **guilhotinado** se, quando aplicado em um retângulo, produz dois novos retângulos. Caso contrário, o corte é do tipo não-guilhotinado. Um padrão é do tipo guilhotinado se é obtido por sucessivos cortes do tipo guilhotinado. A Figura 2.1(a) ilustra um padrão guilhotinado. O padrão é do tipo não-guilhotinado se é obtido por sucessivos cortes guilhotinados e/ou não-guilhotinados.

Um corte é do tipo **não-guilhotinado de primeira ordem** se, quando aplicado em um retângulo, produz cinco novos retângulos arranjados de modo a não formarem um padrão guilhotinado. Um padrão é do tipo não-guilhotinado de primeira ordem se é obtido por sucessivos cortes guilhotinados e/ou não-guilhotinados de primeira ordem. A Figura 2.1(b) ilustra um padrão não-guilhotinado de primeira ordem. Um empacotamento é dito **ortogonal** se as caixas estão dispostas de forma



(a) Padrão guilhotinado. (b) Padrão não-guilhotinado.

Figura 2.1: Padrões de corte.

que um de seus lados seja paralelo a um dos lados do palete.

Dizemos que uma caixa (l, w), com $w \leq l$, tem orientação horizontal se ela estiver disposta no palete (L, W) de forma que seu lado l seja paralelo ao lado L do palete e dizemos que uma caixa tem orientação vertical se seu lado l for paralelo ao lado W do palete. A Figura 2.2 ilustra as duas possíveis orientações.

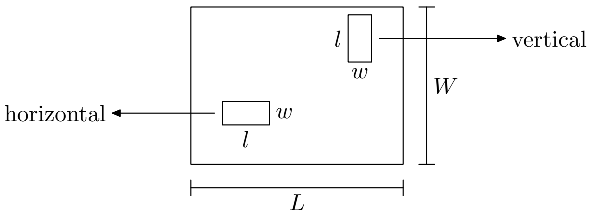


Figura 2.2: Orientações das caixas no retângulo: horizontal e vertical.

Um empacotamento de um retângulo é dito **homogêneo** se todas as caixas arranjadas neste retângulo têm a mesma orientação, como exemplifica a Figura 2.3.

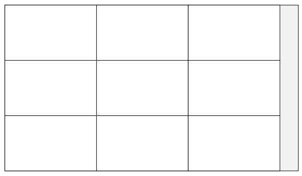


Figura 2.3: Exemplo de empacotamento homogêneo de caixas de dimensões $(5, 3)$ num retângulo de dimensões $(16, 9)$.

2.2 Notações

$\mathbb{Z}_+$ denota o conjunto de números inteiros maiores ou iguais a zero. $\lfloor x \rfloor$ denota o maior inteiro menor ou igual a x. L e W denotam respectivamente o comprimento e a largura do palete. l e w denotam respectivamente o comprimento e a largura das caixas a serem empacotadas. A cardinalidade de um conjunto S é denotada por $|S|$.

Capítulo 3

Algoritmo de cortes não-guilhotinados de primeira ordem

Diversas heurísticas de blocos (retângulos nos quais as caixas empacotadas possuem a mesma orientação) foram propostas para resolver o problema de carregamento de palete do produtor. Entre elas podemos citar as heurísticas de Steudel [17], De Cani e Smith [7], Bischoff e Dowsland [6], Scheithauer e Terno [16] e de Nelissen [15].

Steudel [17] propôs uma heurística na qual o palete é dividido em, no máximo, quatro blocos, onde as caixas são arranjadas segundo uma orientação pré-determinada. Um procedimento semelhante foi proposto por De Cani e Smith [7]. No entanto, este examina uma quantidade muito maior de padrões do que o primeiro.

Bischoff e Dowsland [6] apresentaram um refinamento para os métodos de Steudel [17] e de De Cani e Smith [7], permitindo padrões de empacotamento com um bloco a mais. Nesta heurística, o palete é dividido em, no máximo, cinco blocos, como ilustra a Figura 3.1, e cada um dos blocos tem a orientação das caixas pré-determinada (exceto a orientação das caixas do bloco central que é determinada *a posteriori*, de forma a maximizar o número de caixas). O método baseia-se no fato de que as dimensões do bloco central são determinadas pelas dimensões dos quatro blocos externos e, assim, uma série de padões guilhotinados e não-guilhotinados de primeira ordem são examinados através da variação das dimensões dos blocos externos. O método devolve o padrão que apresenta o maior número de caixas empacotadas.

O método descrito a seguir foi apresentado em [14], por Morabito e Morales, como uma extensão da heurística proposta por Bischoff e Dowsland [6], com a

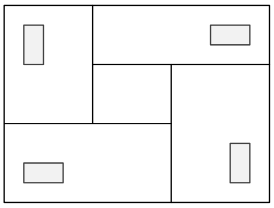


Figura 3.1: A divisão em cinco blocos exibindo a orientação das caixas. A orientação das caixas no bloco central é escolhida de forma a maximizar o número de caixas.

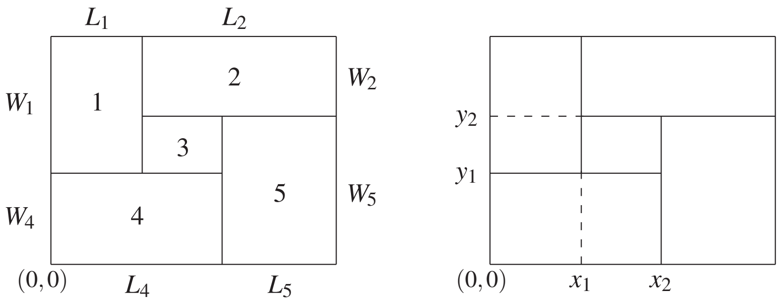
intenção de torná-la mais geral. A idéia é aplicar o método recursivamente em cada bloco gerado e não apenas ao palete. Além disso, a orientação das caixas nos blocos não é previamente determinada. A orientação, em cada bloco, é escolhida de forma a obter um empacotamento com o maior número de caixas possíveis no bloco.

3.1 Descrição do método

Primeiramente, são calculados os limitantes inferior (z_{lb}) e superior (z_{ub}) para o número de caixas (l, w) que podem ser colocadas dentro do retângulo (L, W). Tais limitantes (descritos com mais detalhes na Seção 5) são dados por

$$\begin{aligned} z_{lb} &= \max\left\{ \left\lfloor \frac{L}{l} \right\rfloor \left\lfloor \frac{W}{w} \right\rfloor, \left\lfloor \frac{L}{w} \right\rfloor \left\lfloor \frac{W}{l} \right\rfloor \right\} \text{ e} \\ z_{ub} &= \left\lfloor \frac{L^*W^*}{lw} \right\rfloor, \end{aligned}$$

respectivamente, onde $L^* = \max\{x \mid x = rl + sw,\ x \leq L,\ r, s \in \mathbb{Z}_+\}$ e $W^* = \max\{y \mid y = tw + ul,\ y \leq W,\ t, u \in \mathbb{Z}_+\}$. Se $z_{lb} = z_{ub}$, então a solução ótima foi encontrada e o método devolve z_{lb}. Caso contrário, para cada $x_1, x_2 \in S_L$ tais que $x_1 \leq x_2$ e para cada $y_1, y_2 \in S_W$ tais que $y_1 \leq y_2$, o retângulo (L, W) é dividido em, no máximo, cinco partes (através de cortes guilhotinados ou não-guilhotinados de primeira ordem), numeradas de 1 a 5, com tamanhos (L_i, W_i), conforme ilustra a Figura 3.2(a). Assim, cada divisão fica determinada pela quádrupla (x_1, x_2, y_1, y_2), de acordo com



(a) Tamanho das divisões.

(b) Pontos que determinam o particionamento.

Figura 3.2: Divisão do retângulo em cinco partes, através de um corte não-guilhotinado de primeira ordem.

a Figura 3.2(b), e os tamanhos das partições geradas são dadas por:

$$
\begin{aligned}
(L_1, W_1) &= (x_1, W - y_1) \\
(L_2, W_2) &= (L - x_1, W - y_2) \\
(L_3, W_3) &= (x_2 - x_1, y_2 - y_1) \\
(L_4, W_4) &= (x_2, y_1) \\
(L_5, W_5) &= (L - x_2, y_2)
\end{aligned}
$$

O algoritmo é então aplicado recursivamente a cada uma das partes geradas.

Esse procedimento pode ser representado como uma árvore de busca, na qual o nó raiz representa o problema principal e cada um dos outros nós corresponde a um subproblema do problema representado pelo nó pai. Como o método é aplicado a cada um dos subproblemas recursivamente, na ordem em que são gerados, o procedimento realiza uma busca pela árvore utilizando a estratégia de busca em profundidade. Para evitar de fazer buscas muito profundas, existe a possibilidade de limitar a profundidade da busca em um determinado valor N. Quando a profundidade máxima é alcançada, o método não é aplicado aos subproblemas gerados e cada um deles continua com sua melhor solução obtida até o momento. Um pseudo-código do algoritmo é apresentado a seguir.

3.2 Grade de pontos

Precisamos determinar quais os pontos (x_1, x_2, y_1, y_2) que devemos considerar a fim de explorar o maior número possível de padrões guilhotinados e não-guilhotinados de primeira-ordem. Sejam (x, y) as coordenadas de um vértice de

Algoritmo 1: Pseudo-código do algoritmo descrito em [14].

Entrada: $L, W, l, w, n \in \mathbb{Z}$.

Saída: Número máximo de caixas (l, w) empacotadas no retângulo (L, W).

```
BD-RECURSIVO(L, W, l, w, n)
início
    Determine os conjuntos S_L e S_W para (L, W)
    z_{lb} ← LIMITANTEINFERIOR(L, W, l, w)
    z_{ub} ← LIMITANTESUPERIOR(L, W, l, w)
    se z_{lb} = z_{ub} então
        devolve z_{lb}
    senão
        para cada x_1 ∈ S_L faça
            para cada x_2 ∈ S_L tal que x_1 ≤ x_2 faça
                para cada y_1 ∈ S_W faça
                    para cada y_2 ∈ S_W tal que y_1 ≤ y_2 faça
                        se o padrão produzido não é simétrico a outro então
                            para i ← 1 até 5 faça
                                Determine (L_i, W_i)
                                z^i_{lb} ← LIMITANTEINFERIOR(L_i, W_i, l, w)
                                z^i_{ub} ← LIMITANTESUPERIOR(L_i, W_i, l, w)
                            S_{lb} ← \sum_{i=1}^{5} z^i_{lb}
                            S_{ub} ← \sum_{i=1}^{5} z^i_{ub}
                            se n < N então
                                para i ← 1 até 5 faça
                                    z_i ← BD-RECURSIVO(L_i, W_i, l, w, n+1)
                                    S_{lb} ← z_i − z^i_{lb}
                                    S_{up} ← z_i − z^i_{ub}
                                    se z_{lb} ≥ S_{ub} então
                                        pare
                                    se S_{lb} > z_{lb} então
                                        z_{lb} ← S_{lb}
                                        se z_{lb} = z_{ub} então
                                            devolve z_{lb}
                            se S_{lb} > z_{lb} então
                                z_{lb} ← S_{lb}
        devolve z_{lb}
fim
```

uma caixa (l,w) colocada dentro do retângulo (L,W) (consideramos que o vértice inferior esquerdo do retângulo (L,W) está sobre o ponto $(0,0)$). Assim, x e y podem tomar valores nos conjuntos $G_x = \{x \mid 0 \leq x \leq L\}$ e $G_y = \{y \mid 0 \leq y \leq W\}$, respectivamente. No entanto, podemos substituir esses conjuntos por outros de menor cardinalidade.

Considere a seqüência estritamente crescente $\{z^k\}$ de combinações lineares de l e w, cujos coeficientes são inteiros não negativos.

Teorema 1 *Todo empacotamento ortogonal de caixas (l,w) em um retângulo (L,W) pode ser transformado em outro empacotamento ortogonal, com o mesmo número de caixas, no qual cada vértice (x,y) de cada caixa é tal que $(x,y) = (z^i, z^j)$.*

De acordo com esse teorema, cuja prova pode ser encontrada em [13], podemos reduzir os conjuntos G_x e G_y aos conjuntos normais

$$S(L,l,w) = \{x \mid x = rl + sw,\ x \leq L,\ r,s \in \mathbb{Z}_+\} \text{ e} \tag{3.1}$$
$$S(W,l,w) = \{y \mid y = tw + ul,\ y \leq W,\ t,u \in \mathbb{Z}_+\}. \tag{3.2}$$

Para simplificar, denotaremos $S(L,l,w)$ por S_L e $S(W,l,w)$ por S_W.

A prova do Teorema 1 nos fornece um algoritmo para transformar um empacotamento ortogonal qualquer em outro empacotamento ortogonal no qual cada vértice (x,y) de cada caixa satisfaça $x \in S_L$ e $y \in S_W$. Basta deslocar as caixas, recursivamente, para a esquerda e para baixo enquanto possível sem provocar intersecções.

Note que se o vértice inferior esquerdo de uma caixa está em $S_L \times S_W$ então os demais vértices desta caixa também estão em $S_L \times S_W$.

Sejam $v^0 = (x,y) \in S_L \times S_W$ as coordenadas do vértice inferior esquerdo de uma caixa de dimensões (l,w) colocada num retângulo (L,W). Sejam v^1, v^2 e v^3 as coordenadas dos vértices inferior direito, superior direito e superior esquerdo, respectivamente. Como $(x,y) \in S_L \times S_W$, (x,y) é da forma $(rl+sw, tw+ul)$, com $r,s,t,u \in \mathbb{Z}_+$. Se a orientação da caixa for horizontal, então

$$\begin{aligned} v^1 &= ((r+1)l + sw, tw + ul) \\ v^2 &= ((r+1)l + sw, (t+1)w + ul) \\ v^3 &= (rl + sw, (t+1)w + ul). \end{aligned}$$

Como a caixa está posicionada dentro do retângulo e considerando sua orientação horizontal, temos que $x = rl + sw \leq L - l$ e $y = tw + ul \leq W - w$. Logo,

$$\begin{aligned} v^1_x &= (r+1)l + sw = (rl + sw) + l \leq (L-l) + l = L \\ v^2_y &= (t+1)w + ul = (tw + ul) + w \leq (W-w) + w = W \end{aligned}$$

o que implica em $v_x^1 \in S_L$ e $v_y^2 \in S_W$. Então, $v_x^2 = v_x^1 \in S_L$, $v_y^3 = v_y^2 \in S_W$, $v_y^1 = v_y^0 \in S_W$ e $v_x^3 = v_x^0 \in S_L$ e, portanto, todos os vértices da caixa estão em $S_L \times S_W$. Analogamente obtemos o mesmo resultado se a orientação da caixa for vertical.

3.2.1 *Raster points*

Visando reduzir ainda mais a grade de pontos sobre o retângulo (L,W), foram definidos os conjuntos de *raster points* (cf. [16])

$$\begin{aligned} \widetilde{S}_L &= \{\langle L-x\rangle_{S_L} \mid x \in S_L\} \cup \{0\}, & (3.3)\\ \widetilde{S}_W &= \{\langle L-y\rangle_{S_W} \mid y \in S_W\} \cup \{0\}, & (3.4) \end{aligned}$$

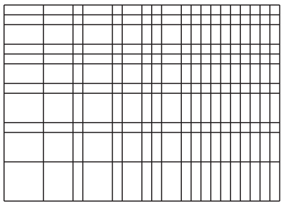
onde

$$\begin{aligned} \langle x'\rangle_{S_L} &= \max\ \{x \in S_L \mid x \leq x'\},\\ \langle y'\rangle_{S_W} &= \max\ \{y \in S_W \mid y \leq y'\} \end{aligned}$$

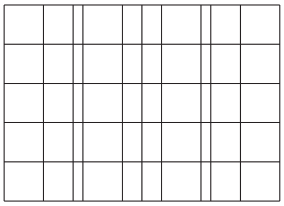
e S_L e S_W são os conjuntos definidos em (3.1) e (3.2), respectivamente. Considere, por exemplo, o problema $(28,20,7,4)$. Os conjuntos normais e *raster points* para esse problema são

$$\begin{aligned} S_L &= \{0,4,7,8,11,12,14,15,16,18,19,20,21,22,23,24,25,26,27,28\},\\ \widetilde{S}_L &= \{0,4,7,8,12,14,16,20,21,24,28\},\\ S_W &= \{0,4,7,8,11,12,14,15,16,18,19,20\} \text{ e}\\ \widetilde{S}_W &= \{0,4,8,12,16,20\}. \end{aligned}$$

Note a redução da cardinalidade dos conjuntos normais para os conjuntos de *raster points*. A Figura 3.3 ilustra a a grade de pontos proporcionada pela utilização dos conjuntos S_L e S_W e pelos conjuntos $\widetilde{S}_L$ e $\widetilde{S}_W$.



(a) Utilizando os conjuntos S_L e S_W.



(b) Utilizando os conjuntos $\widetilde{S}_L$ e $\widetilde{S}_W$.

Figura 3.3: Grade de pontos para $(L,W,l,w) = (28,20,7,4)$.

O teorema a seguir garante que podemos utilizar os conjuntos de *raster points* sem perda de generalidade.

Teorema 2 *Não há perda de generalidade com a utilização dos raster points ao invés dos conjuntos normais.*

Prova. Vamos provar que qualquer empacotamento ortogonal de um retângulo (L,W) no qual cada vértice (x,y) das caixas (l,w) está em $S_L \times S_W$ pode ser transformado em um empacotamento, com o mesmo número de caixas, no qual (x,y) está em $\widetilde{S}_L \times \widetilde{S}_W$.

Considere um empacotamento ortogonal de caixas (l,w) em um retângulo (L,W) em que cada vértice (x,y) de cada caixa é tal que $x \in S_L$ e $y \in S_W$.

Note que se trocarmos a posição x de cada vértice de cada caixa por $x' := L - x$ e cada posição y por $y' := W - y$ continuamos a ter um empacotamento ortogonal sem sobreposição. Podemos pensar nisso como uma reflexão horizontal seguida de uma reflexão vertical das caixas no retângulo.

Pelo Teorema 1, este novo empacotamento pode ser transformado em outro empacotamento ortogonal, com o mesmo número de caixas, no qual cada vértice (x',y') de cada caixa é tal que $(x',y') \in S_L \times S_W$. Isso é obtido deslocando-se cada caixa, recursivamente, para a esquerda e para baixo, sem sobreposições, até que cada vértice satisfaça $(x',y') \in S_L \times S_W$. Podemos fazer isso deslocando as caixas para a posição mais próxima, que é dada por $(\langle x' \rangle_{S_L}, \langle y' \rangle_{S_W})$. Pela definição de *raster points*, temos que $\langle x' \rangle_{S_L} \in \widetilde{S}_L$ e $\langle y' \rangle_{S_W} \in \widetilde{S}_W$. □

3.2.2 Normalização

Um resultado imediato do Teorema 1 é que podemos supor, sem perda de generalidade, que L e W são combinações lineares de l e w com coeficientes inteiros positivos. Assim, o problema (L,W,l,w) é equivalente ao problema $(\bar{L},\bar{W},l,w)$, onde $\bar{L} = \langle L \rangle_{S_L}$ e $\bar{W} = \langle W \rangle_{S_W}$.

3.3 Divisão do palete

Nesta seção exploramos a divisão do retângulo em 2, 3, 4 e 5 partes. Em cada uma delas, optamos por listar todos os casos possíveis como forma de garantir que não estamos perdendo nenhum tipo de divisão. Em seguida, fazemos uma análise das simetrias envolvidas nessas divisões, com o objetivo de reduzir o número de possibilidades de particionamento do retângulo.

3.3.1 Divisão em 2 blocos

Claramente, existem duas formas de dividir um retângulo em dois novos retângulos. Uma delas é fazendo-se um corte horizontal e a outra através de um corte vertical, conforme ilustra a Figura 3.4.

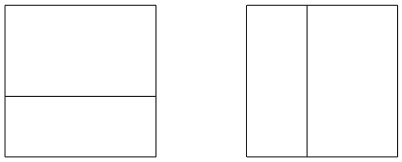


Figura 3.4: Divisão do retângulo em duas partes.

Para realizarmos uma divisão horizontal, é necessário que apenas um corte horizontal atravesse o retângulo do lado esquerdo ao lado direito. Ou seja, uma das três condições a seguir deve ser satisfeita:

1. $y_1 = 0$ e $0 < y_2 < W$
2. $0 < y_1 = y_2 < W$
3. $0 < y_1 < W$ e $y_2 = W$

Além disso, x_1 e x_2 devem tomar valores apenas em $\{0, L\}$ (senão, haveria algum corte vertical e não teríamos uma divisão horizontal em dois blocos), respeitando-se a condição $x_1 \leq x_2$.

Na Tabela 3.1 estão listadas todas as possibilidades de divisão horizontal de um retângulo em dois blocos.

$y_1, y_2 \in Y$	$x_1, x_2 \in X$	Blocos obtidos
$y_1 = 0$ e $0 < y_2 < W$	$x_1 = x_2 = 0$	2 e 5
	$x_1 = 0$ e $x_2 = L$	2 e 3
	$x_1 = x_2 = L$	1
$0 < y_1 = y_2 < W$	$x_1 = x_2 = 0$	2 e 5
	$x_1 = 0$ e $x_2 = L$	3 e 4
	$x_1 = x_2 = L$	1 e 4
$0 < y_1 < W$ e $y_2 = W$	$x_1 = x_2 = 0$	5
	$x_1 = 0$ e $x_2 = L$	3 e 4
	$x_1 = x_2 = L$	1 e 4

Tabela 3.1: Divisões horizontais.

Analogamente, para dividir o retângulo verticalmente, y_1 e y_2 devem tomar valores em $\{0, W\}$, com $y_1 \leq y_2$ e uma das três condições a seguir deve ser satisfeita:

1. $x_1 = 0$ e $0 < x_2 < L$

2. $0 < x_1 = x_2 < L$

3. $0 < x_1 < L$ e $x_2 = L$

Na Tabela 3.2 estão listadas cada uma das possibilidades.

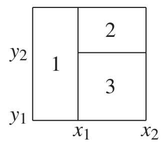
$x_1, x_2 \in X$	$y_1, y_2 \in Y$	Blocos obtidos
$x_1 = 0$ e $0 < x_2 < L$	$y_1 = y_2 = 0$	2
	$y_1 = 0$ e $y_2 = W$	3 e 5
	$y_1 = y_2 = W$	4 e 5
$0 < x_1 = x_2 < L$	$y_1 = y_2 = 0$	1 e 2
	$y_1 = 0$ e $y_2 = W$	1 e 5
	$y_1 = y_2 = W$	4 e 5
$0 < x_1 < L$ e $x_2 = L$	$y_1 = y_2 = 0$	1 e 2
	$y_1 = 0$ e $y_2 = W$	1 e 3
	$y_1 = y_2 = W$	4

Tabela 3.2: Divisões verticais.

3.3.2 Divisão em 3 blocos

Existem $\binom{5}{3} = 10$ diferentes formas de dividirmos um retângulo em três blocos, de acordo com o esquema adotado na Figura 3.1. A seguir são listadas cada uma delas.

- Blocos 1, 2 e 3.
  $0 < x_1 < L$, $x_2 = L$, $y_1 = 0$ e $0 < y_2 < W$.



Prova:
Como queremos os blocos de números 1, 2 e 3, suas dimensões devem ser todas positivas

$$0 < x_1 \tag{3.5}$$
$$0 < W - y_1 \tag{3.6}$$
$$0 < L - x_1 \tag{3.7}$$
$$0 < W - y_2 \tag{3.8}$$
$$0 < x_2 - x_1 \tag{3.9}$$
$$0 < y_2 - y_1 \tag{3.10}$$

e, além disso, as áreas dos blocos de números 4 e 5 devem ser nulas:

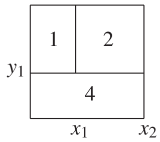
$$x_2 = 0 \text{ ou } y_1 = 0 \tag{3.11}$$
$$L - x_2 = 0 \text{ ou } y_2 = 0 \tag{3.12}$$

Se $x_2 = 0$, então $x_1 = 0$, já que $0 \leq x_1 \leq x_2$. Mas isso viola a condição (3.5). Logo, devemos ter $y_1 = 0$, que satisfaz todas as condições.

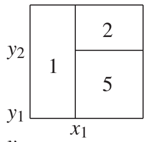
Se $L - x_2 = 0$, ou seja, se $x_2 = L$, todas as condições são satisfeitas. Se $y_2 = 0$, então $y_1 = 0$ (já que $0 \leq y_1 \leq y_2$), o que viola a condição (3.10). Logo, devemos ter necessariamente $x_2 = L$.

Portanto, para obtermos apenas os blocos de números 1, 2 e 3, devemos ter $0 < x_1 < L$, $x_2 = L$, $y_1 = 0$ e $0 < y_2 < W$.

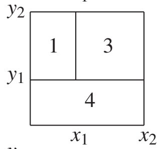
- Blocos 1, 2 e 4.
  $0 < x_1 < L$, $x_2 = L$, $0 < y_1 < W$ e $y_2 = y_1$.



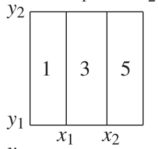
- Blocos 1, 2 e 5.
  $0 < x_1 < L$, $x_2 = x_1$, $y_1 = 0$ e $0 < y_2 < W$.



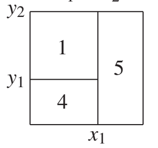
- Blocos 1, 3 e 4.
  $0 < x_1 < L$, $x_2 = L$, $0 < y_1 < W$ e $y_2 = W$.



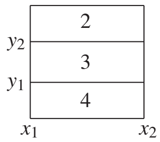
- Blocos 1, 3 e 5.
  $0 < x_1 < x_2 < L$, $y_1 = 0$ e $y_2 = W$.



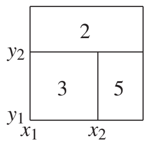
- Blocos 1, 4 e 5.
  $0 < x_1 < L$, $x_2 = x_1$, $0 < y_1 < W$ e $y_2 = W$.



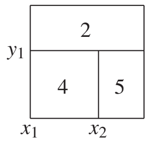
- Blocos 2, 3 e 4.
  $x_1 = 0$, $x_2 = L$ e $0 < y_1 < y_2 < W$.



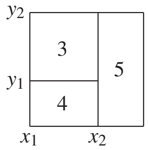
- Blocos 2, 3 e 5.
  $x_1 = 0$, $0 < x_2 < L$, $y_1 = 0$ e $0 < y_2 < W$.



- Blocos 2, 4 e 5.
  $x_1 = 0$, $0 < x_2 < L$, $0 < y_1 < W$ e $y_2 = y_1$.



- Blocos 3, 4 e 5.
  $x_1 = 0$, $0 < x_2 < L$, $0 < y_1 < W$ e $y_2 = W$.



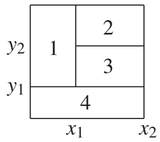
Note, porém, que todos os padrões obtidos na divisão do retângulo em três blocos podem ser obtidos através de dois cortes guilhotinados. O primeiro padrão, por exemplo, no qual permanecem os blocos de números 1, 2 e 3, poderia ser obtido por meio de um corte vertical seguido de um corte horizontal no bloco do lado direito.

Tal fato nos permite ignorar divisões explícitas em três blocos e podemos nos preocupar apenas com divisões em dois blocos.

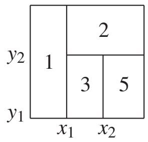
3.3.3 Divisão em 4 blocos

Existem $\binom{5}{4} = 5$ diferentes formas de dividirmos um retângulo em quatro blocos, de acordo com o esquema adotado na Figura 3.1. Essas cinco possibilidades são listadas abaixo.

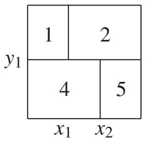
- Blocos 1, 2, 3 e 4.
  $0 < x_1 < L$, $x_2 = L$ e $0 < y_1 < y_2 < W$.

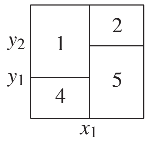


- Blocos 1, 2, 3 e 5.
  $0 < x_1 < x_2 < L$, $y_1 = 0$ e $0 < y_2 < W$.

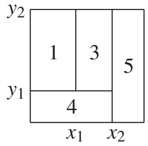


- Blocos 1, 2, 4 e 5.
  ($0 < x_1 \leq x_2 < L$, $0 < y_1 < W$ e $y_2 = y_1$) ou
  ($0 < x_1 < L$, $x_2 = x_1$ e $0 < y_1 \leq y_2 < W$).

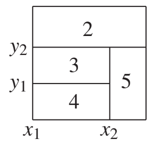




- Blocos 1, 3, 4 e 5.
  $0 < x_1 < x_2 < L$, $0 < y_1 < W$ e $y_2 = W$.



- Blocos 2, 3, 4 e 5.
  $x_1 = 0$, $0 < x_2 < L$ e $0 < y_1 < y_2 < W$.



Da mesma forma ocorrida com as divisões em três blocos, os padrões obtidos com as divisões em quatro blocos também podem ser gerados a partir de combinações de cortes guilhotinados.

3.3.4 Divisão em 5 blocos

Há apenas uma maneira de dividirmos o retângulo em 5 blocos, do modo como o definimos, com x_1, x_2, y_1 e y_2 satisfazendo as seguintes condições: $0 < x_1 < x_2 < L$ e $0 < y_1 < y_2 < W$. A Figura 3.5 ilustra essa divisão.

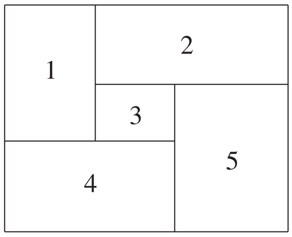


Figura 3.5: Divisão do retângulo em 5 blocos.

3.4 Análise de simetrias

A fim de evitar chamadas recursivas desnecessárias, podemos restringir a busca para um número reduzido de padrões, eliminando particionamentos equivalentes. Como foi mostrado na Seção 3.3, as divisões em três e quatro blocos podem ser derivadas de sucessivos cortes guilhotinados. Assim, analisaremos as simetrias apenas para as divisões em dois e cinco blocos.

Definição 1 *Dizemos que dois particionamentos de um retângulo são equivalentes se eles geram os mesmos subproblemas.*

O Lema a seguir mostra um resultado importante a respeito dos *raster points*, que será muito utilizado daqui em diante.

Lema 1 *Para todo* $x' \in \widetilde{S}_L$, $\langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L} = x'$.

Prova. Seja $x' \in \widetilde{S}_L$. A prova será realizada em duas partes: pimeiro mostraremos que $x' \leq \langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L}$ e, em seguida, mostraremos que $\langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L} \leq x'$.

Temos,

$$\langle L - x' \rangle_{S_L} \leq L - x' \iff x' \leq L - \langle L - x' \rangle_{S_L}.$$

Por definição, $\langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L} = \max A$, onde

$$A = \{ r \in S_L \mid r \leq L - \langle L - x' \rangle_{S_L} \}.$$

Como $x' \leq L - \langle L - x' \rangle_{S_L}$, então $x' \in A$. Logo,

$$x' \leq \max A = \langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L}.$$

Como $x' \in \widetilde{S}_L$, temos que $x' = \langle L - w \rangle_{S_L}$ para algum $w \in S_L$. Além disso, $\langle L - x' \rangle_{S_L} = \max B$, onde

$$B = \{ r \in S_L \mid r \leq L - x' \}.$$

Note que $x' = \langle L - w \rangle_{S_L} \leq L - w$ e, portanto, $w \leq L - x'$. Logo, $w \in B$ e então $w \leq \max B = \langle L - x' \rangle_{S_L}$. Assim, $L - \langle L - x' \rangle_{S_L} \leq L - w$ e isso implica em

$$\langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L} \leq \langle L - w \rangle_{S_L} = x'.$$

Logo, $\langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L} \leq x'$.

Portanto, como $x' \leq \langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L}$ e $\langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L} \leq x'$, temos que $\langle L - \langle L - x' \rangle_{S_L} \rangle_{S_L} = x'$. □

3.4.1 Simetria 2-blocos

Sejam $x_1, x_2 \in \widetilde{S}_L$ e $y_1, y_2 \in \widetilde{S}_W$. No caso da divisão vertical em dois blocos, podemos fixar $y_1 = y_2 = 0$ e $x_2 = x_1$ e fazer x_1 variar no intervalo $(0, \lfloor L/2 \rfloor]$. Dessa forma, evitamos casos simétricos, como mostra o Teorema 3. Analogamente para divisões horizontais, podemos fixar $x_1 = x_2 = 0$ e $y_2 = y_1$ e fazer y_1 variar no intervalo $(0, \lfloor W/2 \rfloor]$. Assim, a busca por padrões pode limitar-se aos dois casos seguintes:

1. $0 < x_1 = x_2 \leq \lfloor L/2 \rfloor$ e $y_1 = y_2 = 0$.
2. $x_1 = x_2 = 0$ e $0 < y_1 = y_2 \leq \lfloor W/2 \rfloor$.

Teorema 3 *Todo particionamento gerado através de um corte guilhotinado em um ponto $x \in \widetilde{S}_L$ tal que $x > \frac{L}{2}$ é equivalente a um particionamento gerado por um corte guilhotinado em um ponto $\bar{x} \in \widetilde{S}_L$ tal que $\bar{x} \leq \frac{L}{2}$.*

Prova. Considere $x \in \widetilde{S}_L$ tal que $x > \frac{L}{2}$. Vamos mostrar que existe $\bar{x} \in \widetilde{S}_L$ tal que $\bar{x} \leq \frac{L}{2}$ e que as partições geradas através do corte em $\bar{x}$ são equivalentes às partições geradas pelo corte em x.

Ao dividirmos o retângulo (L,W) em x, produzimos dois novos retângulos, digamos R_1 e R_2, de tamanhos (x,W) e $(L-x,W)$, respectivamente. Essa divisão é ilustrada na Figura 3.6. O segundo retângulo, ao ser normalizado, passará a ter tamanho $(\bar{x},W)$, onde $\bar{x} = \langle L-x\rangle_{S_L}$. Note que $\bar{x} \in S_L$ e $\bar{x} \leq \lfloor L/2 \rfloor$.

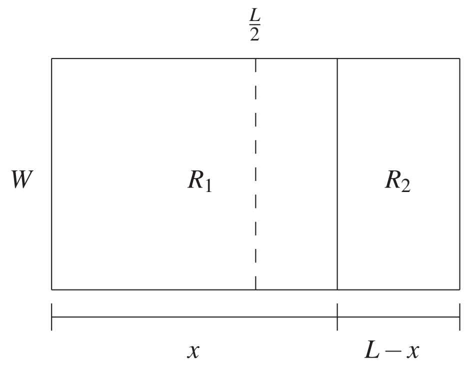


Figura 3.6: Divisão do retângulo (L,W) no ponto x.

Se dividirmos o retângulo no ponto $\bar{x}$, produziremos dois retângulos, digamos $\bar{R}_1$ e $\bar{R}_2$, de tamanhos $(\bar{x},W)$ e $(L-\bar{x},W)$, respectivamente. Normalizando o retângulo $\bar{R}_2$, este terá tamanho $(\langle L-\bar{x}\rangle_{S_L},W)$. Como $\bar{x} = \langle L-x\rangle_{S_L}$, então $\langle L-\bar{x}\rangle_{S_L} = \langle L-\langle L-x\rangle_{S_L}\rangle_{S_L}$ e, pelo Lema 1, $\langle L-\langle L-x\rangle_{S_L}\rangle_{S_L} = x$. Assim, $\bar{R}_1 = (\bar{x},W) = R_2$ e $\bar{R}_2 = (x,W) = R_1$. Ou seja, a divisão do retângulo (L,W) no ponto $\bar{x} = \langle L-x\rangle_{S_L}$ é equivalente à divisão no ponto x. □

3.4.2 Simetria 5-blocos

Analisaremos, agora, as simetrias provenientes de particionamentos do retângulo em 5-blocos através de cortes não-guilhotinados de primeira ordem. Considere uma divisão do retângulo (L,W) em quatro regiões, denominadas A, B, C e D, como mostra a Figura 3.7.

Lema 2 *Para cada corte não-guilhotinado de primeira ordem do retângulo (L,W), determinado por raster points, no qual o centro do bloco de número 3 está no quadrante D (C), existe um particionamento equivalente, dado por outro corte não-guilhotinado de primeira ordem, também determinado por raster points, no qual o centro do bloco 3 está localizado no quadrante A (B).*

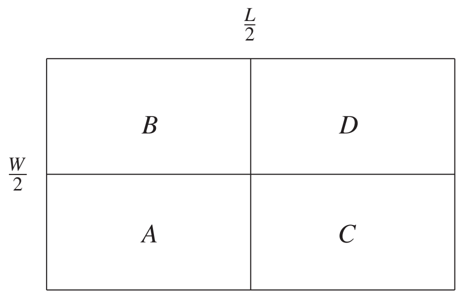


Figura 3.7: Divisão do retângulo em quatro quadrantes.

Prova. Queremos mostrar que cada particionamento gerado através de um corte não-guilhotinado de primeira ordem determinado por *raster points*, que possui o centro do bloco de número 3 $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ no quadrante D (ou C) é equivalente a um outro, cujo centro do bloco 3 está localizado no quadrante A (ou B). Nesse contexto, dois particionamentos são equivalentes quando existe uma bijeção entre os cinco retângulos gerados por cada um deles. Sejam (L,W) as dimensões do retângulo, (l,w) as dimensões das caixas e $\widetilde{S}_L$ e $\widetilde{S}_W$ os conjuntos de *raster points* definidos em (3.3) e (3.4).

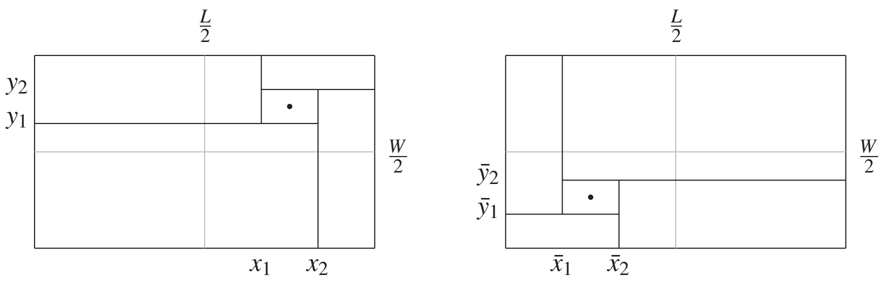
Sejam

$$
\begin{aligned}
(L_1,W_1) &= (x_1, W-y_1) \equiv (x_1, \langle W-y_1\rangle_{S_L}) \\
(L_2,W_2) &= (L-x_1, W-y_2) \equiv (\langle L-x_1\rangle_{S_L}, \langle W-y_2\rangle_{S_L}) \\
(L_3,W_3) &= (x_2-x_1, y_2-y_1) \equiv (\langle x_2-x_1\rangle_{S_L}, \langle y_2-y_1\rangle_{S_L}) \\
(L_4,W_4) &= (x_2, y_1) \\
(L_5,W_5) &= (L-x_2, y_2) \equiv (\langle L-x_2\rangle_{S_L}, y_2)
\end{aligned}
$$

as partições geradas no retângulo (L,W) através de um corte não-guilhotinado de primeira ordem nos pontos $x_1, x_2 \in \widetilde{S}_L$ e $y_1, y_2 \in \widetilde{S}_W$, tais que $\frac{x_1+x_2}{2} > \frac{L}{2}$ e $\frac{y_1+y_2}{2} > \frac{W}{2}$, ou seja, $x_1+x_2 > L$ e $y_1+y_2 > W$.

Considere agora um particionamento nos pontos $\bar{x}_1$, $\bar{x}_2$, $\bar{y}_1$, $\bar{y}_2$, onde

$$
\begin{aligned}
\bar{x}_1 &= \langle L-x_2\rangle_{S_L} \\
\bar{x}_2 &= \langle L-x_1\rangle_{S_L} \\
\bar{y}_1 &= \langle W-y_2\rangle_{S_L} \\
\bar{y}_2 &= \langle W-y_1\rangle_{S_L}.
\end{aligned}
$$



(a) Bloco 3 localizado no quadrante *D*.

(b) Bloco 3 localizado no quadrante *A*.

Figura 3.8: Em (a), particionamento nos pontos x_1, x_2, y_1, y_2, e em (b), nos pontos $\bar{x}_1, \bar{x}_2, \bar{y}_1, \bar{y}_2$.

Pela definição de *raster points*, $\bar{x}_1, \bar{x}_2 \in \widetilde{S}_L$ e $\bar{y}_1, \bar{y}_2 \in \widetilde{S}_W$. Assim,

$$
\begin{aligned}
(\bar{L}_1, \bar{W}_1) &= (\bar{x}_1, \langle W - \bar{y}_1 \rangle_{S_L}) = (\langle L - x_2 \rangle_{S_L}, \langle W - \langle W - y_2 \rangle_{S_L} \rangle_{S_L}) \\
&= (\langle L - x_2 \rangle_{S_L}, y_2) = (L_5, W_5) \\
(\bar{L}_2, \bar{W}_2) &= (\langle L - \bar{x}_1 \rangle_{S_L}, \langle W - \bar{y}_2 \rangle_{S_L}) \\
&= (\langle L - \langle L - x_2 \rangle_{S_L} \rangle_{S_L}, \langle W - \langle W - y_1 \rangle_{S_L} \rangle_{S_L}) = (x_2, y_1) = (L_4, W_4) \\
(\bar{L}_4, \bar{W}_4) &= (\bar{x}_2, \bar{y}_1) = (\langle L - x_1 \rangle_{S_L}, \langle W - y_2 \rangle_{S_L}) = (L_2, W_2) \\
(\bar{L}_5, \bar{W}_5) &= (\langle L - \bar{x}_2 \rangle_{S_L}, \bar{y}_2) = (\langle L - \langle L - x_1 \rangle_{S_L} \rangle_{S_L}, \langle W - y_1 \rangle_{S_L}) \\
&= (x_1, \langle W - y_1 \rangle_{S_L})) = (L_1, W_1) \\
(\bar{L}_3, \bar{W}_3) &= (\langle L - \bar{L}_5 - \bar{L}_1 \rangle_{S_L}, \langle W - \bar{W}_4 - \bar{W}_2 \rangle_{S_L}) \\
&= (\langle L - L_1 - L_5 \rangle_{S_L}, \langle W - W_2 - W_4 \rangle_{S_L}) = (\langle L_3 \rangle_{S_L}, \langle W_3 \rangle_{S_L}) \\
&= (L_3, W_3)
\end{aligned}
$$

Logo, $(\bar{L}_1, \bar{W}_1) = (L_5, W_5)$, $(\bar{L}_2, \bar{W}_2) = (L_4, W_4)$, $(\bar{L}_3, \bar{W}_3) = (L_3, W_3)$, $(\bar{L}_4, \bar{W}_4) = (L_2, W_2)$ e $(\bar{L}_5, \bar{W}_5) = (L_1, W_1)$ e, portanto, o particionamento determinado pela quádrupla (x_1, x_2, y_1, y_2), que possui o centro do bloco 3 no quadrante *D*, é equivalente ao obtido por $(\bar{x}_1, \bar{x}_2, \bar{y}_1, \bar{y}_2)$, que possui o centro do bloco 3 no quadrante *A*.

Analogamente, cada particionamento que possui o centro do bloco 3 no quadrante *C* é equivalente a um particionamento cujo centro do bloco 3 está no quadrante *B*. $\square$

Enunciaremos e provaremos, a seguir, o Teorema que garante que as simetrias envolvidas em cortes não-guilhotinados de primeira ordem, utilizadas em [14], continuam válidas com a utilização dos *raster points*.

Teorema 4 *A busca por padrões não guilhotinados de primeira ordem pode limitar-se apenas aos casos em que* $x_1, x_2 \in \widetilde{S}_L$ *e* $y_1, y_2 \in \widetilde{S}_W$*, com* $0 < x_1 < x_2 < L$ *e* $0 < y_1 < y_2 < W$*, são tais que*

$$\begin{aligned} &x_1 + x_2 < L, \text{ ou} \\ &x_1 + x_2 = L \text{ e } y_1 + y_2 \leq W. \end{aligned}$$

Prova. Sejam $x_1, x_2 \in \widetilde{S}_L$ e $y_1, y_2 \in \widetilde{S}_W$ tais que $0 < x_1 < x_2 < L$ e $0 < y_1 < y_2 < W$ e considere o particionamento do retângulo (L, W) por meio de um corte não-guilhotinado de primeira ordem determinado por esses pontos. Pelo Lema 2, o centro do bloco 3, dado por $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$, não precisa ocupar as regiões C e D. Assim, a coordenada x do centro não precisa ultrapassar $\frac{L}{2}$, ou seja $\frac{x_1+x_2}{2} \leq \frac{L}{2}$, o que implica em

$$x_1 + x_2 \leq L. \tag{3.13}$$

A condição (3.13) pode ser dividida em duas partes:

$$\begin{cases} x_1 + x_2 = L \\ x_1 + x_2 < L \end{cases}$$

Se $x_1 + x_2 = L$, então, pelo Lema 2, a coordenada y do centro do bloco 3 não precisa ultrapassar $\frac{W}{2}$, ou seja, $\frac{y_1+y_2}{2} \leq \frac{W}{2}$, o que nos leva a

$$y_1 + y_2 \leq W.$$

□

Porém, isso não é suficiente para eliminar todos os casos simétricos. Quando $L = W$, também são válidas simetrias entre as regiões A e B, de modo que um padrão não-guilhotinado de primeira ordem gerado com o centro do bloco 3 na região B possui um padrão equivalente no qual o centro do bloco 3 está na região A. Assim, quando o retângulo é um quadrado, a busca por padrões não-guilhotinados de primeira ordem pode limitar-se aos casos em que $x_1 + x_2 \leq L$ e $y_1 + y_2 \leq W$.

3.5 Detalhes de implementação

Nesta seção apresentamos os detalhes mais importantes de nossa implementação. São eles:

- Incorporação dos *raster points*.

- Geração apenas dos padrões não simétricos entre si.
- Eliminação de divisões do retângulo em 3 e 4 partes explicitamente. Agora elas são geradas por sucessivos cortes guilhotinados ao longo da recursão.
- Tratamento de subproblemas repetidos.
- Armazenamento de informações.
- Cálculo dos limitantes.

3.5.1 Incorporação dos *raster points*

Em nossa implementação, utilizamos os conjuntos de *raster points* (3.3) e (3.4), ao invés dos conjuntos normais (3.1) e (3.2), para formar a grade de particionamento do retângulo. Houve uma grande melhora com a incorporação desses conjuntos e os resultados obtidos podem ser vistos na Seção 3.6.

3.5.2 Geração de padrões

Com os resultados obtidos em nossa análise de simetrias, apresentada na Seção 3.4, decidimos modificar a forma de gerar as partições. No algoritmo original [14], são geradas todas as divisões possíveis do retângulo e a cada novo particionamento é verificado se o padrão produzido é simétrico a algum outro, para decidir se aquele padrão precisa ser considerado ou não. Ao invés disso, decidimos fazer com que o algoritmo produzisse apenas os padrões não simétricos entre si, economizando processamento.

Isso foi feito substituindo os quatro laços encaixados que geram os pontos x_1, x_2, y_1 e y_2 (ver Algoritmo 1), por três conjuntos de laços separadamente:

(a) um bloco de laços encaixados para gerar particionamentos em cinco retângulos;

para cada $x_1 \in \widetilde{S}_L$ tal que $x_1 \leq \lfloor \frac{L}{2} \rfloor$ **faça**
  para cada $x_2 \in \widetilde{S}_L$ tal que $x_1 < x_2 < L$ **faça**
    para cada $y_1 \in \widetilde{S}_W$ tal que $y_1 < W$ **faça**
      para cada $y_2 \in \widetilde{S}_W$ tal que $y_1 < y_2 < W$ **faça**
        ⋮

(b) um laço para divisões horizontais;

para cada $x_1 \in \widetilde{S}_L$ tal que $x_1 \leq \lfloor \frac{L}{2} \rfloor$ **faça**
$x_2 \leftarrow x_1$
$y_1 \leftarrow 0$
$y_2 \leftarrow 0$
$\vdots$

(c) e outro para divisões verticais.

para cada $y_1 \in \widetilde{S}_W$ tal que $y_1 \leq \lfloor \frac{W}{2} \rfloor$ **faça**
$y_2 \leftarrow y_1$
$x_1 \leftarrow 0$
$x_2 \leftarrow 0$
$\vdots$

Ainda no laço (a), se $x_1 + x_2 = L$, então podemos restringir y_1 e y_2 apenas aos casos em que $y_1 + y_2 \leq W$.

Com essa separação em três laços, as divisões explícitas do retângulo em três e quatro partes foram eliminadas e agora são obtidas por sucessivos cortes guilhotinados nos laços (b) e (c). Porém, essa alteração implica numa mudança no conceito de profundidade máxima que se tinha inicialmente. Antes, com profundidade igual a 1 já era possível obter padrões com 3 e 4 blocos, enquanto que agora tais padrões só aparecerão nas profundidades 2 e 3. Isso pode fazer com que alguns problemas que podiam ser resolvidos com profundidade máxima $N = 3$ necessitem agora de uma profundidade máxima um pouco maior para serem resolvidos.

3.5.3 Subproblemas repetidos

Ao longo da execução do algoritmo um mesmo subproblema pode surgir mais de uma vez. Antes de resolver qualquer subproblema, é preciso verificar se ele já foi resolvido e assim evitar processamento desnecessário. Porém, não podemos simplesmente ignorar um subproblema repetido e devolver o valor de sua solução anterior.

Como a profundidade da recursão (ou da árvore de busca) é limitada, um subproblema pode não ser completamente explorado, no sentido de que a busca limita-se a um número reduzido de padrões, de forma que não lhe seja atribuída a melhor solução possível. Uma solução melhor poderia ser encontrada caso a profundidade da busca fosse um pouco maior.

Com isso em mente, ao nos depararmos com um subproblema repetido verificamos em que profundidade ele apareceu anteriormente. Por definição, a profundidade de um subproblema ainda não resolvido é N (profundidade máxima permitida) e a profundidade de um problema resolvido de forma comprovadamente ótima é 0. Caso tenha sido numa profundidade menor ou igual do que a atual, devolvemos a solução anterior, já que se o subproblema fosse resolvido a partir da profundidade atual, a qualidade da solução encontrada seria, no melhor caso, tão boa quanto a obtida anteriormente. Caso ele tenha surgido em uma profundidade maior do que a atual, "damos mais uma chance" ao subproblema e o resolvemos novamente, já que existe a possibilidade de encontrarmos uma solução melhor.

No caso em que $N = \infty$, não há necessidade de resolver um mesmo subproblema novamente caso ele venha a aparecer em uma profundidade menor do que a anterior. Pois, como não há limite para a profundidade da busca, todos os padrões possíveis são examinados e a melhor solução possível é encontrada. Assim, cada subproblema é resolvido uma única vez. Além disso, podemos reduzir o limitante superior para o valor da solução encontrada, já que sabemos que não é possível encontrar uma solução melhor com essa abordagem.

3.5.4 Armazenamento de informações

As informações dos subproblemas são armazenadas em uma matriz bidimensional de tamanho $|\widetilde{S_L}||\widetilde{S_W}|$, no caso da utilização dos *raster points*, ou $|S_L||S_W|$, no caso da utilização dos conjuntos normais.

São utilizados um vetor de tamanho L para indexar a primeira coordenada da matriz e um vetor de tamanho W para indexar a segunda coordenada da matriz. Assim, para encontrar a posição da matriz em que é armazenada a informação de um subproblema com retângulo de comprimento L_s e largura W_s (sempre considerando $L_s \geq W_s$) basta acessar a posição L_s do primeiro vetor, obtendo um índice i, e a posição W_s do segundo vetor, obtendo um índice j, de forma que o subproblema em questão é armazenado na posição (i, j) da matriz.

Essa estrutura possibilita armazenar e recuperar a informação de um subproblema em tempo constante e sua exigência de memória é $O(|\widetilde{S_L}||\widetilde{S_W}| + L)$.

3.5.5 Cálculo dos limitantes

O cálculo dos limitantes inferiores e superiores de todos os subproblemas é feito previamente e eles são armazenados na estrutura descrita na Seção 3.5.4. Isso mostrou-se mais eficiente do que o cálculo do limitante apenas no momento de sua utilização.

3.6 Experimentos

Os testes foram realizados em um computador com processador AMD Athlon 64 3200+ 2202.87 MHz, com 1 GB de memória RAM e sistema operacional Linux. Fizemos nossa implementação do algoritmo em linguagem C e compilamos o programa utilizando o GCC (GNU Compiler Collection, versão 3.3.5) com a opção -O4 de otimização (opção máxima de otimização).

A implementação original do algoritmo foi gentilmente cedida pelos autores de [14], e assim pudemos compará-la com a nossa implementação. Ela foi feita em Pascal e a compilamos com o GPC (GNU Pascal Compiler) com a opção -O3 de otimização (opção máxima de otimização).

Testamos os algoritmos com conjuntos de instâncias bem conhecidos na literatura [9, 1], chamados *Cover I* e *Cover II*. *Cover I* é o conjunto de classes de equivalências de instâncias (L, W, l, w) satisfazendo

$$1 \leq \frac{L}{W} \leq 2, \qquad 1 \leq \frac{l}{w} \leq 4, \qquad 1 \leq \frac{LW}{lw} < 51$$

e *Cover II* o conjunto de classes de equivalências de instâncias satisfazendo

$$1 \leq \frac{L}{W} \leq 2, \qquad 1 \leq \frac{l}{w} \leq 4, \qquad 51 \leq \frac{LW}{lw} < 101.$$

Cover I tem 7827 classes de equivalências, enquanto que *Cover II* possui 40609. Morabito e Morales [14] restringiram seus testes às instâncias com $L, W \leq 1000$, totalizando 3179 instâncias de *Cover I* (que chamaremos de *Cover* I_B) e 16938 instâncias de *Cover II* (que chamaremos de *Cover* II_B) e são elas que utilizamos em nossos testes.

A Tabela 3.3 mostra os resultados obtidos para a execução do algoritmo com as instâncias de *Cover* II_B para diferentes valores de profundidade máxima (primeira coluna). A segunda coluna exibe os tempos, em segundos, gastos pela implementação original e a terceira coluna os tempos gastos por nossa implementação. A última coluna mostra a relação entre os dois.

Com $N = 3$, o algoritmo original encontrou a solução ótima de todas as instâncias de *Cover* I_B e não encontrou a solução ótima de apenas 16 instâncias de *Cover* II_B (que mesmo com $N = \infty$ não puderam ser encontradas). Os valores encontrados por nossa implementação foram os mesmos.

A instância que consumiu mais tempo de processamento da implementação original foi (86, 68, 15, 4), que levou 27.00 segundos para ser resolvida. Já a nossa implementação, para esta mesma instância, demorou apenas 0.23 segundo.

3.6.1 Análise individual

Para calcular o ganho proporcionado pelas principais modificações realizadas no algoritmo, retiramos de nossa implementação cada uma delas e comparamos com a versão completa do algoritmo. Os principais elementos que diferenciam nossa implementação da implementação original são:

N	Tempo (segundos) Implementação original (A)	Nossa implementação (B)	A / B
3	5238.67	73.08	71.68
4	6665.58	75.73	88.01
5	7814.75	78.54	99.50
6	8619.68	80.38	107.24
7	9144.11	81.29	112.49
8	9449.20	81.56	115.86
9	9659.74	82.55	117.02
10	9711.25	82.91	117.13
∞	9845.40	71.16	138.36

Tabela 3.3: Tempo gasto, em segundos, para resolver todos os problemas de *Cover* II_B. Apenas 16 problemas não foram resolvidos de forma ótima.

- Grade de pontos formada por *raster points*.
- Geração apenas dos padrões não simétricos entre si.
- Utilização do limitante de Barnes. (Os experimentos são mostrados na Seção 5.3.)
- Indexação eficiente da matriz.

Os dados das tabelas a seguir referem-se à resolução de todos os problemas do conjunto *Cover* II_B.

A Tabela 3.4 mostra a relação entre a implementação completa e a implementação que utiliza os conjuntos normais ao invés dos *raster points* para formar a grade de pontos. Podemos ver que implementação que utiliza os *raster points* é quase quatro vezes mais rápida do que a versão que utiliza os conjuntos normais.

A Tabela 3.5 apresenta uma comparação entre a versão completa do algoritmo e uma versão em que os pontos de corte x_1, x_2, y_1 e y_2 gerados acabam produzindo padrões simétricos que são detectados através de uma verificação posterior. Na implementação final, os padrões simétricos simplesmente não são produzidos.

A Tabela 3.6 mostra uma comparação entre uma versão com indexação ineficiente da matriz de informações e a implementação completa. Isso é apenas um detalhe que não foi tratado na implementação original do algoritmo mas que faz uma grande diferença. Os índices associados a um determinado subproblema não eram acessados diretamente (conforme explicado na Seção 3.5.4) e fazia-se uma busca em um vetor para encontrá-los.

N	Tempo (segundos)		A / B
	Sem *raster points* (A)	Implementação completa (B)	
3	246.13	73.08	3.36
4	263.16	75.73	3.47
5	281.21	78.54	3.58
6	291.03	80.38	3.62
7	296.42	81.29	3.64
8	298.72	81.56	3.66
9	299.32	82.55	3.62
10	300.10	82.91	3.61
∞	244.09	71.16	3.43

Tabela 3.4: Comparação entre a versão completa e a versão sem *raster points* (implementada com os conjuntos normais).

N	Tempo (segundos)		A / B
	Sem a separação dos laços (A)	Implementação completa (B)	
3	110.82	73.08	1.51
4	111.81	75.73	1.47
5	117.45	78.54	1.49
6	124.19	80.38	1.54
7	130.97	81.29	1.61
8	136.66	81.56	1.67
9	139.88	82.55	1.69
10	142.35	82.91	1.71
∞	84.05	71.16	1.18

Tabela 3.5: Comparação entre a versão completa e a versão que produz os padrões simétricos.

N	Tempo (segundos)		A / B
	Indexação ineficiente (A)	Implementação completa (B)	
3	254.77	73.08	3.48
4	257.08	75.73	3.39
5	264.13	78.54	3.36
6	268.82	80.38	3.34
7	270.59	81.29	3.32
8	271.09	81.56	3.32
9	272.08	82.55	3.29
10	271.91	82.91	3.27
∞	243.16	71.16	3.41

Tabela 3.6: Comparação entre a implementação completa do algoritmo e a versão com indexação ineficiente da matriz de informações.

Capítulo 4

Algoritmo-L

Em [13], Lins, Lins e Morabito propuseram substituir o particionamento recursivo em cinco regiões retangulares, descrito no Capítulo 3, por um particionamento recursivo de um retângulo (que chamaremos de R) ou de uma peça em forma de L (que chamaremos de L) em duas peças, cada uma das quais sendo um R ou um L. Essa forma de particionar o retângulo é mais geral do que a anterior, já que realiza todas as divisões efetuadas pelo Algoritmo 1, entre elas os cortes não-guilhotinados de primeira ordem, e ainda divide um retângulo em peças em forma de L.

Um L é determinado por quatro inteiros (X,Y,x,y) e seu posicionamento padrão é definido como o fechamento topológico do retângulo cuja diagonal vai de $(0,0)$ até (X,Y) menos o retângulo que vai de (x,y) até (X,Y), com $X \geq x$ e $Y \geq y$. Quando $x = X$ e $y = Y$, $L(X,Y,x,y) = L(X,Y,X,Y)$ é o retângulo que vai de $(0,0)$ até (X,Y) e pode ser considerado como um L degenerado. $L(X,Y,x,y)$ denota o retângulo determinado por $(X,Y,x,y) \in \mathbb{Z}_+^4$ e $R(X,Y)$ denota o L degenerado $L(X,Y,X,Y)$.

Os autores afirmaram que há sete formas de dividir um L em dois $L's$, sendo cinco delas a partir de um L e duas a partir de um R, que são mostradas na Figura 4.1. Cada subdivisão é denotada por B_k, $k \in \{1,\ldots,7\}$. No entanto, encontramos mais duas formas, ilustradas na Figura 4.2, não relacionadas em [13]. Essas novas divisões são consideradas daqui em diante e serão denotadas por B_8 e B_9.

4.1 Descrição do método

Cada subdivisão B_k, $k \in \{1,2,3,4,5,8,9\}$, de um $L(X,Y,x,y)$ em dois novos $L's$, é determinada por dois parâmetros internos, (x',y'), que indicam o local onde a divisão é realizada. O conjunto de tais parâmetros é denotado por $P_k'(L)$. Se

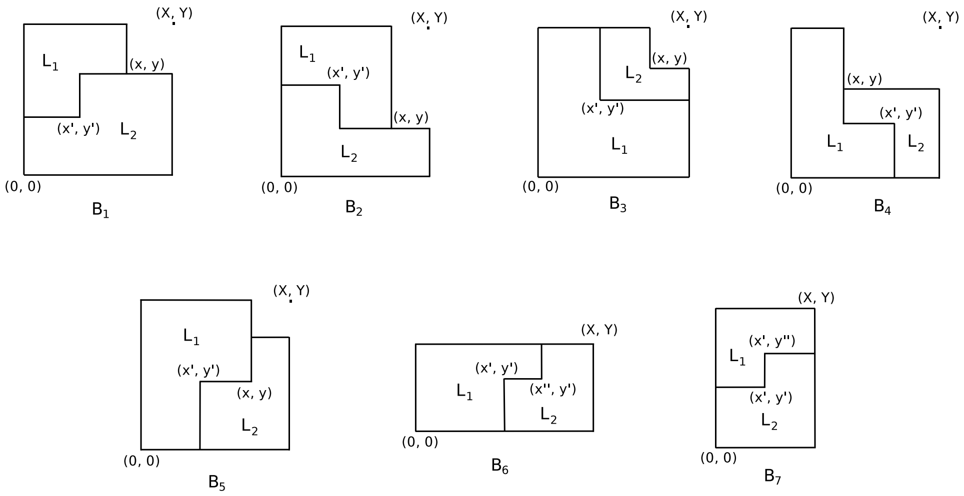


Figura 4.1: Subdivisões B_1, B_2, B_3, B_4 e B_5 de um L em dois L's e B_6 e B_7 de um R em dois L's.

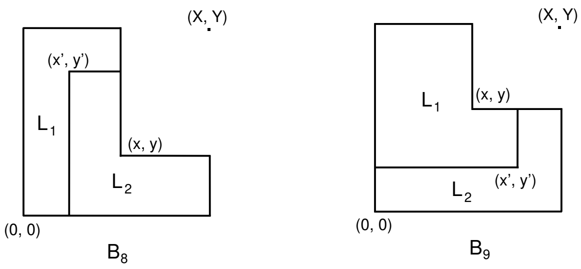


Figura 4.2: Duas novas formas de particionar uma peça em L em dois novos L's, não previstas em [13].

$k \in \{1,3,5\}$, então

$$P'_k(L) = \{p_k = (x',y') \mid 0 \leq x' \leq x,\ 0 \leq y' \leq y\},$$

se $k \in \{2,8\}$, então

$$P'_k(L) = \{p_k = (x',y') \mid 0 \leq x' \leq x,\ y \leq y' \leq Y\},$$

e se $k \in \{4,9\}$, temos

$$P'_k(L) = \{p_k = (x',y') \mid x \leq x' \leq X,\ 0 \leq y' \leq y\}.$$

As subdivisões B_6 e B_7 de um $R(X,Y)$ em dois novos $L's$ são determinadas por três parâmetros internos, e $P_6(R)$ e $P_7(R)$ são dados por:

$$\begin{aligned}
P'_6(R) &= \{p_6 = (x',x'',y') \mid 0 \leq x' \leq x'' \leq X,\ 0 \leq y' \leq Y\} \\
P'_7(R) &= \{p_7 = (x',y',y'') \mid 0 \leq x' \leq X,\ 0 \leq y' \leq y'' \leq Y\}.
\end{aligned}$$

Os dois novos $L's$ gerados na subdivisão B_k, $k \in \{1,\dots 9\}$, são denotados por $\mathscr{L}_i(L,k,p_k)$, para $i \in \{1,2\}$, e são dados por:

$$\begin{aligned}
B_1\!:\ & \mathscr{L}_1(L,1,p_1) = (x,Y-y',x',Y-y) \\
& \mathscr{L}_2(L,1,p_1) = (X,y,X-x',y')
\end{aligned}$$

$$\begin{aligned}
B_2\!:\ & \mathscr{L}_1(L,2,p_2) = (x,Y-y,x-x',Y-y') \\
& \mathscr{L}_2(L,2,p_2) = (X,y',x',y)
\end{aligned}$$

$$\begin{aligned}
B_3\!:\ & \mathscr{L}_1(L,3,p_3) = (X,Y,x',y') \\
& \mathscr{L}_2(L,3,p_3) = (X-x',Y-y',X-x',y-y')
\end{aligned}$$

$$\begin{aligned}
B_4\!:\ & \mathscr{L}_1(L,4,p_4) = (x',Y,x,y') \\
& \mathscr{L}_2(L,4,p_4) = (X-x,y,X-x',y-y')
\end{aligned}$$

$$\begin{aligned}
B_5\!:\ & \mathscr{L}_1(L,5,p_5) = (x,Y,x',Y-y') \\
& \mathscr{L}_2(L,5,p_5) = (X-x',y,X-x,y')
\end{aligned}$$

$$\begin{aligned}
B_6\!:\ & \mathscr{L}_1(L,6,p_6) = (x'',Y,x',Y-y') \\
& \mathscr{L}_2(L,6,p_6) = (X-x',Y,X-x'',y')
\end{aligned}$$

$$\begin{aligned}
B_7\!:\ & \mathscr{L}_1(L,7,p_7) = (X,Y-y',x',Y-y'') \\
& \mathscr{L}_2(L,7,p_7) = (X,y'',X-x',y')
\end{aligned}$$

$$\begin{aligned}
B_8\!:\ & \mathscr{L}_1(L,8,p_8) = (x,Y,x',Y-y') \\
& \mathscr{L}_2(L,8,p_8) = (X-x',y',x-x',y)
\end{aligned}$$

B_9: $\mathscr{L}_1(L,9,p_9) = (x', Y-y', x, y-y')$
$\quad\;\mathscr{L}_2(L,9,p_9) = (X, y, X-x', y')$

Cada $L = L(X,Y,x,y)$ gerado em uma subdivisão válida deve ter área estritamente positiva, ou seja, $XY - (X-x)(Y-y) > 0$. Assim, os valores de p_k ficam restritos ao subconjunto $P_k(L)$ de $P'_k(L)$, dado por $P_k(L) = \{p_k \in P'_k(L) \mid A(\mathscr{L}_1(L,k,p_k)) > 0$ e $A(\mathscr{L}_2(L,k,p_k)) > 0\ \}$, para $k \in \{1,\dots 9\}$, onde $A(L)$ denota a área de L.

Essa abordagem é baseada numa função $v^\star$ definida recursivamente, cujo domínio é o conjunto de $L's$ válidos e cujo valor de $v^\star(L)$ é o número de caixas (l,w) que podem ser arranjadas em L. Essa função é definida através da seguinte relação de recorrência: Seja $L = L(X,Y,x,y)$. Se a área de L é menor do que a área da caixa (l,w), isto é, se $XY-(X-x)(Y-y) < lw$, ou se $XY-(X-x)(Y-y) = lw$, mas $L \neq R(l,w)$, então não cabe nenhuma caixa em L e definimos $v^\star(L) = 0$. Caso $L = R(l,w)$, então cabe exatamente um caixa em L e definimos $v^\star(L) = 1$. Agora que já está definida a base da recursão, podemos definir $v^\star(L)$ como

$$v^\star(L) = \max_{1\le k\le 9}\{\max_{p_k\in P_k(L)}\{v^\star(\mathscr{L}_1(L,k,p_k)) + v^\star(\mathscr{L}_2(L,k,p_k))\}\}.$$

4.2 Detalhes de implementação

Em nossa implementação, incorporamos os conjuntos de *raster points*, descritos na Seção 3.2.1, e as duas novas formas de divisão de um L em dois novos L's, mostradas na Figura 4.2.

O ponto mais delicado do Algoritmo-L, que pode limitá-lo a resolver apenas problemas não muito grandes, está no armazenamento das informações dos subproblemas. Birgin, Morabito e Nishihara [5] apresentaram uma estrutura de dados para armazenar as informações dos subproblemas.

A quantidade de memória necessária é proporcional a $|S_L|^2|S_W|^2$, onde $|S_L|$ e $|S_W|$ são os tamanhos dos conjuntos S_L e S_W definidos em (3.1) e (3.2), respectivamente.

Porém, com a utilização dos *raster points*, a quantidade de memória necessária para resolver um problema pode ser reduzida para $|\widetilde{S}_L|^2|\widetilde{S}_W|^2$, onde $\widetilde{S}_L$ e $\widetilde{S}_W$ são os conjuntos de *raster points* do problema em questão. Como as dimensões de um subproblema são formadas por (i) *raster points* e/ou (ii) diferenças de *raster points* do problema original, o Teorema 5 mostra que essas dimensões continuam sendo *raster points* do problema original. Além disso, pela definição desses conjuntos, os conjuntos de *raster points* de um subproblema serão subconjuntos dos *raster points* do problema orginal.

Teorema 5 *Se $x \in \widetilde{S}_L$, $y \in S_L$ e $x \geq y$ então $\langle x-y\rangle_{S_L} \in \widetilde{S}_L$. Em particular, $\langle x-y\rangle_{\widetilde{S}_L} = \langle x-y\rangle_{S_L}$, onde*

$$
\begin{aligned}
S_L &= \{rl+sw \mid rl+sw \leq L,\ r,s \in \mathbb{Z}_+\},\\
\langle u\rangle_{S_L} &= \max\{a \in S_L \mid a \leq u\},\\
\widetilde{S}_L &= \{\langle L-u\rangle_{S_L} \mid u \in S_L\} \text{ e}\\
\langle u\rangle_{\widetilde{S}_L} &= \max\{a \in \widetilde{S}_L \mid a \leq u\}.
\end{aligned}
$$

Prova. Como $x \in \widetilde{S}_L$, existe $z \in S_L$ tal que $x = \langle L-z\rangle_{S_L}$. Afirmamos que $\langle x-y\rangle_{S_L} = \langle L-z-y\rangle_{S_L}$. Note que isso implica o teorema pois $\langle L-z-y\rangle_{S_L} = \langle L-(z+y)\rangle_{S_L} \in \widetilde{S}_L$ pois $z+y \in S_L$ já que a soma s de quaisquer dois elementos de S_L pertence a S_L se $s \leq L$. (Note que $z+y \leq L$ pois $L-z-y \geq x-y \geq 0$.)

Para mostrar que $\langle x-y\rangle_{S_L} = \langle L-z-y\rangle_{S_L}$ note que:

(a) $x = \langle L-z\rangle_{S_L} \Longrightarrow x \leq L-z \Longrightarrow x-y \leq L-z-y \Longrightarrow \langle x-y\rangle_{S_L} \leq \langle L-z-y\rangle_{S_L}$.

(b) Por definição de $\langle L-z-y\rangle_{S_L}$, existem r_1 e s_1 tais que

  (i) $r_1 l + s_1 w = \langle L-z-y\rangle_{S_L}$

  (ii) $r_1, s_1 \geq 0$

  (iii) $r_1 l + s_1 w \leq L-z-y$.

  Como y pertence a S_L, existem r_y e $s_y \geq 0$ tais que $y = r_y l + s_y w$. Logo, de (iii) concluímos que $(r_1+r_y)l + (s_1+s_y)w \leq L-z$. Então, como $x = \max\{rl+sw \mid r,s \geq 0,\ rl+sw \leq L-z\}$, temos que $x \geq (r_1+r_y)l + (s_1+s_y)w = r_1 l + s_1 w + y$. Logo, $x-y \geq r_1 l + s_1 w$. Daí, por definição de $\langle x-y\rangle_{S_L}$, concluímos que $\langle x-y\rangle_{S_L} \geq r_1 l + s_1 w$ e (i) mostra que $\langle x-y\rangle_{S_L} \geq \langle L-z-y\rangle_{S_L}$.

Os items (a) e (b) implicam que $\langle x-y\rangle_{S_L} = \langle L-z-y\rangle_{S_L}$ e a prova está completa.
□

4.2.1 Memória

A estrutura de dados usada para armazenar as informações dos subproblemas é uma matriz de quatro dimensões com $|\widetilde{S}_L|^2|\widetilde{S}_W|^2$ entradas. As informações de um subproblema $(X_s, Y_s, x_s, y_s) \in \widetilde{S}_L \times \widetilde{S}_W \times \widetilde{S}_L \times \widetilde{S}_W$ são armazenadas na posição (I, J, i, j), onde I e i são armazenados respectivamente nas posições X_s e x_s de um vetor de tamanho X e J e j são armazenados respectivamente nas posições Y_s e y_s de um vetor de tamanho Y, considerando que as dimensões do retângulo do problema principal são (X, Y). Dessa forma, o armazenamento e recuperação de informações de um subproblema são realizados em tempo constante.

Porém, para alguns problemas maiores, $|\widetilde{S}_L|^2|\widetilde{S}_W|^2$ se torna muito grande de forma que não é possível alocar a quantidade memória necessária para essa estrutura. De acordo com os experimentos mostrados em [5], a quantidade de memória realmente utilizada pelo algoritmo é muito pequena. Em média, apenas 1.5% dos elementos da matriz são utilizados pelo algoritmo. O ideal seria alocar memória apenas para o que é utilizado. No entanto, não sabemos *a priori* quais posições da matriz serão utilizadas pelo algoritmo (ou seja, quais subproblemas aparecerão durante a execução do algoritmo) e seria necessária uma estrutura que pudesse crescer conforme novos subproblemas fossem aparecendo, como uma lista ligada por exemplo. No entanto, o algoritmo seria prejudicado em termos de eficiência. Existe aí um compromisso entre a quantidade de memória alocada e a velocidade do algoritmo.

A solução encontrada foi combinar uma matriz multidimensional com uma árvore binária de busca. Ao invés de alocar espaço para uma matriz de tamanho $|\widetilde{S}_L|^2|\widetilde{S}_W|^2$, criamos uma matriz tridimensional de tamanho $|\widetilde{S}_L|^2|\widetilde{S}_W|$ em que cada posição possui uma árvore. Assim, na posição (I,J,i), com I, J e i associados a X_s, Y_s e x_s, respectivamente, temos uma árvore que guarda informações de todos os subproblemas da forma $(X_s,Y_s,x_s,\cdot)$. Se não for possível alocar essa quantidade de memória, criamos uma matriz bidimensional de tamanho $|\widetilde{S}_L||\widetilde{S}_W|$ que na posição (I,J) possui uma tabela de árvore que armazena informações a respeito dos subproblemas da forma $(X_s,Y_s,\cdot,\cdot)$. Se essa quantidade de memória também não puder ser alocada, criamos um vetor de tamanho $|\widetilde{S}_L|$ que em cada posição possui árvore que armazena informações dos subproblemas da forma $(X_s,\cdot,\cdot,\cdot)$. Por fim, se ainda não for possível alocar tal quantidade de memória, criamos uma árvore para armazenar todos os subproblemas.

4.2.2 Combinação dos algoritmos

A grande vantagem do algoritmo descrito no Capítulo 3 (Algoritmo 1) é a sua velocidade. Ele é muito mais rápido do que o Algoritmo-L, embora não encontre soluções ótimas em alguns casos nos quais o Algoritmo-L é capaz de encontrar. Sendo assim, o ideal seria um algoritmo que combinasse a velocidade do primeiro com a abrangência do segundo. Então, implementamos uma versão que combina os dois algoritmos da seguinte forma: primeiramente, o Algoritmo 1 é aplicado ao problema que desejamos resolver. Caso não seja possível comprovar a otimalidade da solução encontrada, aplicamos o Algoritmo-L ao problema. Como na maioria dos casos o primeiro algoritmo encontra a solução ótima com garantia de otimalidade, o Algoritmo-L será executado poucas vezes. Além disso, nos casos em que o Algoritmo-L tiver de ser utilizado, todas as informações acumuladas na execução do primeiro algoritmo, entre elas os limitantes e as soluções dos subproblemas, são aproveitadas, poupando ainda mais processamento.

4.3 Experimentos

Os experimentos foram realizados em um computador com processador AMD Athlon 64 3200+ 2202.87 MHz, com 1 GB de memória RAM e sistema operacional Linux. Nossa implementação foi programada em linguagem C e compilamos o programa utilizando o GCC (GNU Compiler Collection, versão 3.3.5) com a opção -O4 de otimização.

Na Tabela 4.1 estão os resultados alcançados por nossa implementação com *raster points* e na Tabela 4.2 estão os resultados obtidos pela implementação proposta em [13], ambas considerando as duas novas formas de divisão do L. As instâncias de problemas são as mesmas que foram utilizadas nos testes descritos na Seção 3.6. Para as instâncias selecionadas de *Cover II*, nossa implementação foi cerca de 12 vezes mais rápida do que a implementação original.

Conjunto de problemas	Número de problemas	Tempo (segundos)				
		Total	Média	Desvio padrão	Mín.	Máx.
16 hard	16	133.59	8.34	5.61	0.49	19.63
Cover I_B	3179	1650.24	0.51	0.33	0.00	2.48
Cover II_B	16938	83298.61	4.91	6.90	0.00	56.91

Tabela 4.1: Nossa implementação com *raster points*.

Conjunto de problemas	Número de problemas	Tempo (segundos)				
		Total	Média	Desvio padrão	Mín.	Máx.
16 hard	16	695.34	43.45	41.23	1.20	155.02
Cover I_B	3179	3187.41	1.00	1.66	0.00	27.36
Cover II_B	16938	944231.50	55.74	106.00	0.00	1479.46

Tabela 4.2: Implementação proposta em [13] com os conjuntos (3.1) e (3.2).

A instância de *Cover* II_B que demorou mais tempo para ser resolvida pela implementação original foi $(L,W,l,w) = (167,83,23,6)$, que levou 1479.46 segundos. Já a nossa implementação, para essa mesma instância, demorou apenas 34.20 segundos para encontrar a solução.

Na Tabela 4.3 estão os resultados obtidos considerando apenas as 16 instâncias mais difíceis, não resolvidas por outras heurísticas, entre elas a proposta em [14], que foi descrita no Capítulo 3. Nossa implementação foi quase 6 vezes mais rápida. As representações gráficas das soluções desses problemas são exibidas no Apêndice A.

Problemas	Tempo (segundos)	
$P_n = (L, W, l, w)$	Sem *raster points*	Com *raster points*
$P_{53} = (43, 26, 7, 3)$	1.20	0.49
$P_{57} = (49, 28, 8, 3)$	3.66	1.48
$P_{69} = (57, 34, 7, 4)$	6.93	3.09
$P_{69} = (63, 44, 8, 5)$	7.30	1.92
$P_{71} = (61, 35, 10, 3)$	18.33	5.21
$P_{75} = (67, 37, 11, 3)$	33.56	8.37
$P_{77} = (61, 38, 10, 3)$	27.75	8.19
$P_{77} = (61, 38, 6, 5)$	9.59	1.37
$P_{81} = (67, 40, 11, 3)$	49.27	12.77
$P_{82} = (74, 49, 11, 4)$	63.39	11.55
$P_{82} = (93, 46, 13, 4)$	79.84	11.29
$P_{96} = (106, 59, 13, 5)$	155.02	13.90
$P_{96} = (141, 71, 13, 8)$	67.53	9.22
$P_{97} = (74, 46, 7, 5)$	27.55	11.24
$P_{99} = (86, 52, 9, 5)$	90.74	19.63
$P_{100} = (108, 65, 10, 7)$	53.68	13.87
Média	43.45	8.34
Total	695.34	133.59

Tabela 4.3: Tempo para resolver 16 problemas não resolvidos por outras heurísticas. A primeira coluna apresenta tais problemas. A coluna do meio mostra o tempo para a implementação do Algoritmo-*L* sem *raster points* (utilizando os conjuntos normais) e a terceira coluna o tempo da implementação com *raster points*.

Com a mudança na estrutura de dados descrita na Seção 4.2.1, foi possível resolver todos os problemas de *Cover III*, proposto recentemente [1], que ainda não haviam sido resolvidos por esse método. O *Cover III* possui instâncias representando 98016 classes de equivalências satisfazendo

$$1 \leq \frac{L}{W} \leq 2, \qquad 1 \leq \frac{l}{w} \leq 4, \qquad 101 \leq \frac{LW}{lw} < 151.$$

Lorena e Ribeiro [12] apresentaram algumas instâncias reais obtidas junto a portos brasileiros para o problema de estivagem de unidades de celulose em porões de navios. Muitas delas apresentam soluções com mais de 200 caixas e também puderam ser resolvidas devido à nova estrutura de dados. As Figuras 4.4 e 4.3 exibem as soluções encontradas pelo algoritmo para duas dessas instâncias.

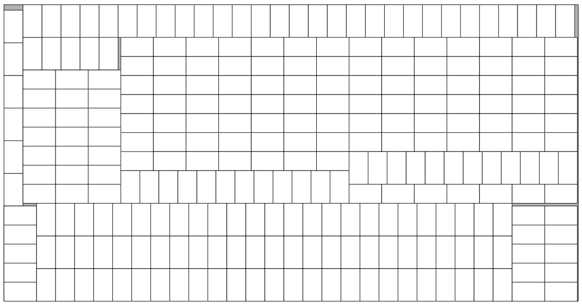


Figura 4.3: Instância $(2536, 1312, 144, 84)$ com 273 caixas.

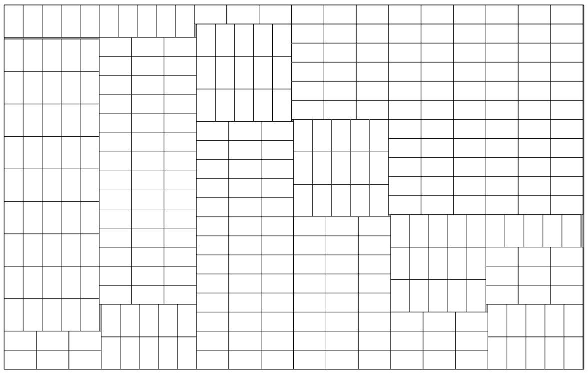


Figura 4.4: Instância $(2560, 1610, 143, 84)$ com 341 caixas.

Capítulo 5

Limitantes

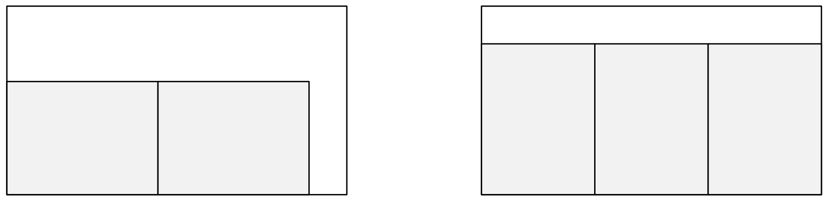
Neste capítulo apresentamos os limitantes utilizados nos algoritmos que estudamos. Como limitantes superiores, foram utilizados o limitante da área e um outro baseado no limitante de Barnes [2]. Na Seção 5.3 são apresentados alguns experimentos que mostram o aumento de desempenho do Algoritmo 1 com a introdução do limitante de Barnes.

5.1 Limitante inferior

O limitante a seguir é baseado no empacotamento homogêneo:

$$\max\left\{\left\lfloor\frac{L}{l}\right\rfloor\left\lfloor\frac{W}{w}\right\rfloor, \left\lfloor\frac{L}{w}\right\rfloor\left\lfloor\frac{W}{l}\right\rfloor\right\} \tag{5.1}$$

O primeiro termo considera o empacotamento de caixas dispostas horizontalmente e o segundo apenas de caixas colocadas verticalmente. A Figura 5.1 exemplifica o cálculo desse limitante.



(a) Empacotamento horizontal. (b) Empacotamento vertical.

Figura 5.1: Exemplo com $(L,W) = (9,5)$ e $(l,w) = (4,3)$. Em (a), foi obtido o valor 2 como limitante inferior, enquanto que em (b) o valor obtido foi 3.

5.2 Limitantes superiores

O limitante superior mais simples é baseado nas áreas do retângulo (L,W) e da caixa (l,w). Apenas é feita a divisão entre a área do retângulo e área da caixa: $\lfloor \frac{LW}{lw} \rfloor$. É possível melhorar um pouco esse limitante utilizando a idéia de normalização apresentada na Seção 3.2.2. Ao invés de calcularmos utilizando a área total do retângulo (L,W), podemos utilizar a área útil:

$$\left\lfloor \frac{L^*W^*}{lw} \right\rfloor \tag{5.2}$$

onde

$$\begin{aligned} L^* &= \max\{x \mid x = rl + sw,\ x \leq L,\ r,s \in \mathbb{Z}_+\} \text{ e} \\ W^* &= \max\{y \mid y = tw + ul,\ y \leq W,\ t,u \in \mathbb{Z}_+\}. \end{aligned}$$

5.2.1 Limitante de Barnes

Em [2], Barnes descreve um procedimento para encontrar o valor exato da área não aproveitada em um empacotamento ótimo de caixas (l,w), com l e w primos entre si, em um retângulo “suficientemente grande”.

Além disso, o resultado encontrado é também um limitante inferior para a área não aproveitada no empacotamento de um retângulo qualquer, de forma que este valor pode ser utilizado para estimar um limitante superior para o número de caixas que podem ser empacotadas em um retângulo.

O procedimento é baseado no fato de que qualquer empacotamento de caixas (l,w) em um retângulo (L,W) é também um empacotamento ou de caixas $(l,1)$ ou de caixas $(1,w)$ em (L,W).

Vamos considerar o empacotamento de caixas $(l,1)$ em um retângulo (L,W), com $l \leq L,W$. Sejam r,s inteiros tais que $L \equiv r \pmod{l}$ e $W \equiv s \pmod{l}$, com $0 \leq r,s < l$. A área A não aproveitada no empacotamento ótimo de caixas $(l,1)$ em (L,W) é dada por

$$A = \begin{cases} rs & \text{se } r+s \leq l \\ (l-r)(l-s) & \text{se } r+s \geq l, \end{cases}$$

de acordo com o Lema 1 em [2]. Note que $A = \min\{rs,(l-r)(l-s)\}$. Analogamente, para o empacotamento de caixas $(1,w)$, a área não aproveitada é $B = \min\{tu,(w-t)(w-u)\}$, onde t,u são inteiros que satisfazem $L \equiv t \pmod{w}$, $W \equiv u \pmod{w}$ e $0 \leq t,u < w$.

Assim, a área desperdiçada D no empacotamento de caixas (l,w) em (L,W) satisfaz:

$$\begin{aligned} D &\geq \max\{A,B\} && (5.3)\\ D &\equiv LW (\text{mod } lw). && (5.4)\end{aligned}$$

Sejam Q_1 e R_1 inteiros tais que $LW = Q_1(lw)+R_1$, com $0 \leq R_1 < lw$ e sejam Q_2 e R_2 inteiros tais que $\max\{A,B\} = Q_2(lw)+R_2$, com $0 \leq R_2 < lw$. Então, se $R_1 \geq R_2$, temos $D = Q_2(lw)+R_1$. Senão, temos D = $(Q_2+1)lw+R_1$.

Como D é um limitante inferior para a área desperdiçada, temos que $LW-D$ é um limitante superior para a área aproveitada e

$$\frac{LW-D}{lw} \tag{5.5}$$

nos dá um limitante superior para o número de caixas (l,w) que podem ser colocadas no retângulo (L,W).

5.3 Experimentos

Realizamos alguns testes com o limitante de Barnes utilizando os problemas descritos nas Seções 3.6 e 4.3. O limitante de Barnes foi melhor do que o limitante (5.2) em torno de 4.2% das vezes, o mesmo valor obtido por Dowsland [8].

Considerando o algoritmo de cortes não-guilhotinados de primeira ordem, apresentado no Capítulo 3, a utilização do limitante superior como sendo o mínimo entre o limitante baseado na área e o limitante de Barnes produziu um aumento de mais de duas vezes na velocidade do algoritmo em comparação com a versão que utiliza apenas o limitante da área. Os resultados são exibidos na Tebela 5.1. Os tempos referem-se à resolução de todos os problemas de *Cover* II_B.

A grande redução de tempo deve-se ao fato de que com um bom limitante superior (e com um bom limitante inferior) é possível detectar mais rapidamente a otimalidade das soluções obtidas e assim interromper a busca por possíveis soluções melhores.

N	Tempo (segundos) A	B	A / B
3	153.21	73.08	2.09
4	176.65	75.73	2.33
5	190.59	78.54	2.42
6	197.12	80.38	2.45
7	200.10	81.29	2.46
8	200.72	81.56	2.46
9	201.58	82.55	2.44
10	202.09	82.91	2.43
∞	146.69	71.16	2.06

Tabela 5.1: A coluna A mostra os tempos obtidos pela implementação que utiliza como limitante superior apenas o limitante (5.2) baseado na área do retângulo. A coluna B mostra os tempos obtidos pela implementação que utiliza como limitante superior o mínimo entre (5.2) e (5.5).

Capítulo 6

Conclusões e trabalhos futuros

Implementamos o algoritmo proposto em [14] incorporando de forma eficiente os *raster points*. Provamos que, com a utilização de *raster points*, as simetrias entre os padrões de cortes continuam válidas e modificamos a forma de gerar as partições, fazendo com que o algoritmo produza apenas os padrões não simétricos. Também eliminamos as divisões explícitas do retângulo em três e quatro partes, que agora são obtidas naturalmente por sucessivos cortes guilhotinados ao longo da recursão. Testes mostraram que a nossa implementação com as modificações apresentadas é, em média, mais de 70 vezes mais rápida do que a implementação original.

Também implementamos o algoritmo proposto em [13], incorporando os *raster points* e duas novas formas de dividir uma peça em L em dois novos L's, que não estavam previstas em [13]. Provamos alguns resultados que mostram que a quantidade de memória exigida para armazenar informações dos subproblemas pode ser ainda menor do que a que foi mostrada em [5]. Experimentos mostraram que a incorporação dos *raster points* fez com que o algoritmo tivesse um desempenho 12 vezes superior ao do algoritmo originalmente proposto em [13].

Além disso, implementamos uma versão que combina os dois algoritmos, reunindo a velocidade do primeiro e a robustez do segundo e implementamos uma versão do Algoritmo-L que utiliza uma estrutura de dados que se adapta ao tamanho do problema, permitindo resolver problemas maiores.

Pretendemos agora estudar as simetrias envolvidas nas divisões em peças em forma de L e aproveitar os resultados para tentar implementar uma versão mais eficiente do algoritmo.

Parte II

Experiência Pessoal

Capítulo 7

A Iniciação Científica e o BCC

Descrevo brevemente neste capítulo a experiência adquirida durante a iniciação científica e o Bacharelado em Ciência da Computação (BCC).

7.1 Desafios e frustrações

Durante as férias que precederam o quarto ano do BCC comecei a desenvolver esse trabalho de iniciação científica. Acho que o início foi o período mais difícil, já que eu não tinha nenhuma familiaridade com o problema abordado.

Após um estudo detalhado dos artigos que descrevem os algoritmos comecei a implementá-los. Depois de alcançarmos uma implementação fiel do algoritmo descrito no artigo [14] começamos a modificá-lo a fim de torná-lo mais eficiente. Nesse momento surgiram novas dificuldades, principalmente em relação às provas dos teoremas que garantem que as modificações propostas preservam a corretude do algoritmo.

Houve muito tempo gasto com os testes dos algoritmos. Isso porque cada alteração realizada no algoritmo implicava em rodar os testes novamente, que já eram demorados. Apesar dos testes não me impedirem de avançar no trabalho, era preciso aguardar o término deles para fazer uma avaliação das mudanças realizadas no algoritmo antes de introduzir novas mudanças.

Na metade do primeiro semestre de 2006, apresentei um seminário para o grupo de Otimização Contínua sobre o trabalho desenvolvido nesse período. Como ainda não possuía experiência nesse tipo de apresentação, o seminário foi um grande desafio para mim.

Uma outra dificuldade foi a redação de relatórios científicos para a FAPESP. Muitas vezes quando lemos determinados artigos deixamos alguns detalhes de lado. No entanto, ao escrevermos uma demonstração, por exemplo, é preciso cuidar desses detalhes para que esta seja completa. Por outro lado, escrever e documentar os avanços aos poucos ajudou bastante na elaboração dos relatórios.

Ao longo do curso não tive muitas frustrações. Uma delas foi a escassez de computadores disponíveis para executar os testes dos algoritmos.

7.2 Escolha do orientador

No final do segundo ano do BCC, eu estava disposto a realizar um trabalho de iniciação científica (IC). A principal motivação foi a minha intenção de continuar meus estudos no mestrado. A iniciação me ajudaria a escolher uma área de estudo e seria uma primeira experiência de pesquisa acadêmica.

Como não tinha muita certeza em que área gostaria de realizá-lo, decidi conversar com diversos professores para saber em que trabalhavam e quais eram as possibilidades para uma iniciação científica. Em particular, como havia gostado das disciplinas MAC315 – Programação Linear e MAC300 – Métodos Numéricos de Álgebra Linear, fui conversar com o Professor Ernesto G. Birgin. Ele me apresentou algumas propostas e me interessei por uma delas. Tratava-se do estudo e da implementação de uma versão paralela do algoritmo de gradientes conjugados.

No final do terceiro ano, após terminar a implementação do algoritmo, o professor Ernesto me apresentou um novo trabalho. Tratava-se de um problema de empacotamento de retângulos em retângulos, que foi o tema desta monografia.

7.3 Interação com o orientador

Desde o primeiro momento ele foi muito atencioso e estava sempre disposto a ajudar. Não havia um dia específico para reuniões, mas procurava me encontrar com ele toda semana. Sempre que eu tinha novidades ou dúvidas eu passava na sala dele para conversarmos.

A orientação não limitava-se apenas ao trabalho de iniciação científica. Ao mesmo tempo que preocupava-se com meus avanços no trabalho, também preocupava-se com meu desempenho nas disciplinas do curso. Além disso, no final dos semestres, ajudava-me no momento de definir as disciplinas optativas que eu iria cursar.

Além do trabalho de iniciação científica, tínhamos um projeto paralelo que reunia outros orientandos do Ernesto. O projeto incluía a criação de interfaces entre o algoritmo para programação não-linear ALGENCAN[1], escrito em Fortran77, e as outras linguagens de programação.

[1] http://www.ime.usp.br/~egbirgin/tango/

7.4 Disciplinas mais relevantes

Considero que as disciplinas mais relevantes para esse trabalho foram MAC122 (Princípios de Desenvolvimento de Algoritmos), MAC323 (Estruturas de Dados), MAC211 (Laboratório de Programação I) e MAC338 (Análise de Algoritmos).

- MAC122 – Princípios de Desenvolvimento de Algoritmos
  Nesta disciplina começamos a estudar estruturas de dados (listas, pilhas e filas) que permitem implementar algoritmos mais sofisticados e começamos a nos preocupar com a eficiência dos algoritmos.

- MAC323 – Estruturas de Dados
  Estudamos com mais detalhes as estruturas de dados introduzidas na disciplina MAC122 e outras estruturas como árvores binárias de busca e árvores balanceadas.

- MAC211 – Laboratório de Programação I
  Nessa disciplina aprendi definitivamente a linguagem C (utilizada, juntamente com C++, na implementação dos algoritmos).

- MAC338 – Análise de Algoritmos
  Imprescindível para quem pretende escrever um algoritmo eficiente. Aprendemos a analisar um algoritmo tanto em relação ao consumo de tempo quanto em relação ao consumo de espaço.

7.5 Considerações finais

Apesar de não termos cumprido totalmente o plano inicial (não abordamos os modelos não-lineares para empacotamento de retângulos em regiões convexas arbitrárias), estou muito satisfeito com o trabalho realizado, visto que concluímos todas as tarefas importantes que planejamos.

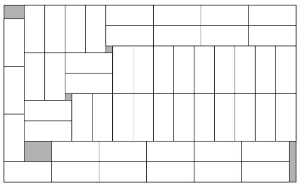
No próximo ano pretendo ingressar no mestrado em Ciência da Computação do IME. O professor Ernesto continuará sendo o meu orientador e estudaremos algoritmos para problemas de programação não-linear inteira mista.

Por fim, agradeço à minha família, que sempre me apoiou e me ajudou, aos grandes amigos, colegas e professores do IME que fizeram parte desses meus quatro anos no BCC.

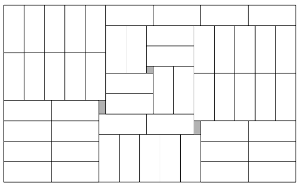
Apêndice A

Soluções das 16 instâncias mais difíceis

Neste apêndice exibimos as representações gráficas das soluções encontradas pelo algoritmo de cortes não-guilhotinados de primeira ordem e pelo Algoritmo *L* para os 16 problemas da Tabela 4.3, apresentada no Capítulo 4. As soluções encontradas pelo Algoritmo-*L* para esses 16 problemas são todas ótimas, enquanto que cada uma das soluções encontradas pelo primeiro algoritmo apresenta um padrão com uma caixa a menos.

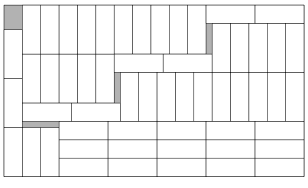


(a) Algoritmo 1: 52 caixas.

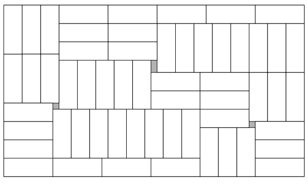


(b) Algoritmo-*L*: 53 caixas.

Figura A.1: Instância $(43, 26, 7, 3)$.

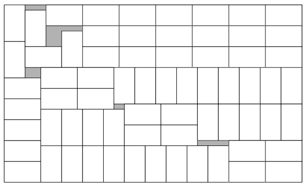


(a) Algoritmo 1: 56 caixas.

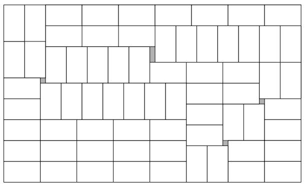


(b) Algoritmo-*L*: 57 caixas.

Figura A.2: Instância $(49, 28, 8, 3)$.

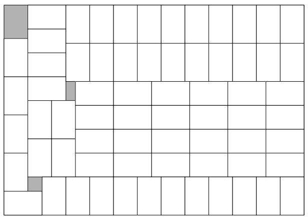


(a) Algoritmo 1: 68 caixas.

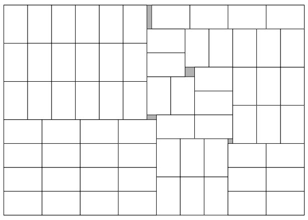


(b) Algoritmo-L: 69 caixas.

Figura A.3: Instância $(57, 34, 7, 4)$.

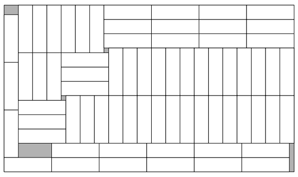


(a) Algoritmo 1: 68 caixas.

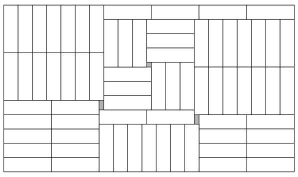


(b) Algoritmo-L: 69 caixas.

Figura A.4: Instância $(63, 44, 8, 5)$.



(a) Algoritmo 1: 70 caixas.

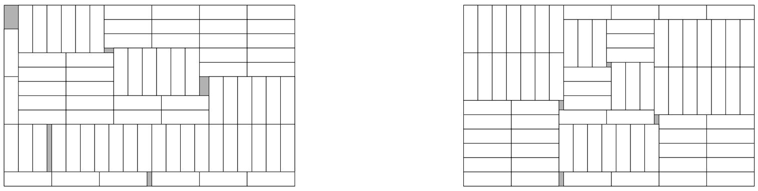


(b) Algoritmo-L: 71 caixas.

Figura A.5: Instância $(61, 35, 10, 3)$.

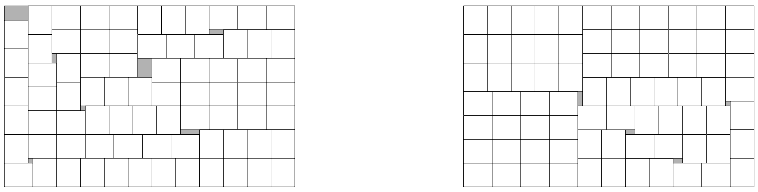
(a) Algoritmo 1: 74 caixas. (b) Algoritmo-*L*: 75 caixas.

Figura A.6: Instância $(67,37,11,3)$.



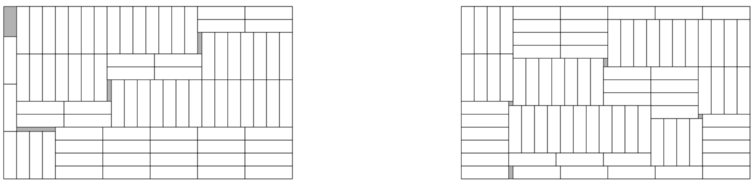
(a) Algoritmo 1: 76 caixas. (b) Algoritmo-*L*: 77 caixas.

Figura A.7: Instância $(61,38,10,3)$.



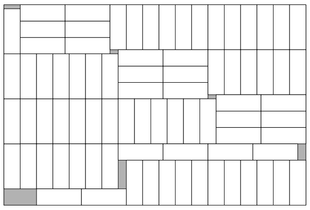
(a) Algoritmo 1: 76 caixas. (b) Algoritmo-*L*: 77 caixas.

Figura A.8: Instância $(61,38,6,5)$.

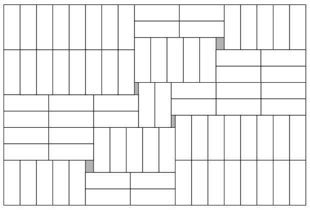


(a) Algoritmo 1: 80 caixas. (b) Algoritmo-*L*: 81 caixas.

Figura A.9: Instância $(67,40,11,3)$.

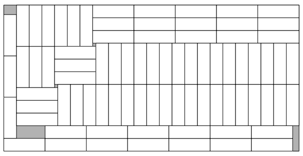


(a) Algoritmo 1: 81 caixas.

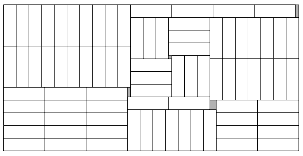


(b) Algoritmo-*L*: 82 caixas.

Figura A.10: Instância $(74, 49, 11, 4)$.

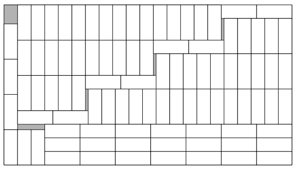


(a) Algoritmo 1: 81 caixas.

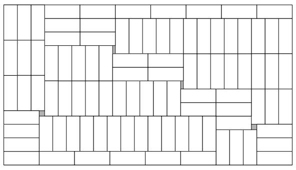


(b) Algoritmo-*L*: 82 caixas.

Figura A.11: Instância $(93, 46, 13, 4)$.

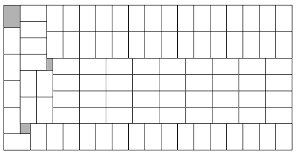


(a) Algoritmo 1: 95 caixas.

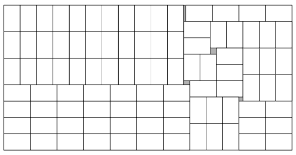


(b) Algoritmo-*L*: 96 caixas.

Figura A.12: Instância $(106, 59, 13, 5)$.

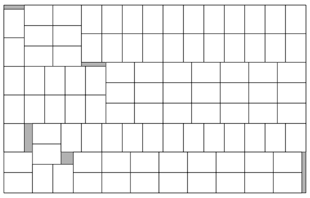


(a) Algoritmo 1: 95 caixas.

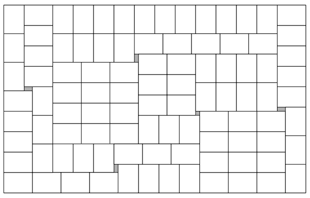


(b) Algoritmo-*L*: 96 caixas.

Figura A.13: Instância $(141, 71, 13, 8)$.

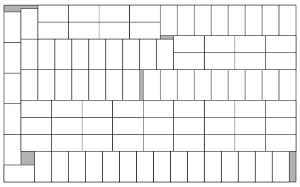


(a) Algoritmo 1: 96 caixas.

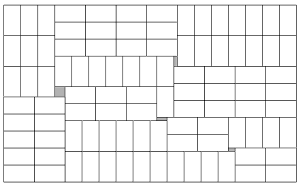


(b) Algoritmo-L: 97 caixas.

Figura A.14: Instância $(74, 46, 7, 5)$.

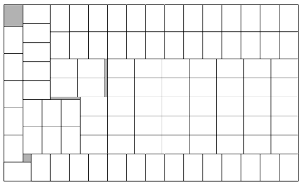


(a) Algoritmo 1: 98 caixas.

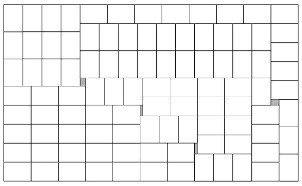


(b) Algoritmo-L: 99 caixas.

Figura A.15: Instância $(86, 52, 9, 5)$.



(a) Algoritmo 1: 99 caixas.



(b) Algoritmo-L: 100 caixas.

Figura A.16: Instância $(108, 65, 10, 7)$.

Referências Bibliográficas

[1] R. Alvarez-Valdes, F. Parreño e J. M. Tamarit. A tabu search algorithm for the pallet loading problem. *OR Spectrum*, 27:43–61, 2005.

[2] F. W. Barnes. Packing the maximum number of $m \times n$ tiles in a large $p \times q$ rectangle. *Discrete Mathematics*, 26:93–100, 1979.

[3] E. G. Birgin, J. M. Martínez, F. H. Nishihara e D. P. Ronconi. Orthogonal packing of rectangular items within arbitrary convex regions by nonlinear optimization. *Computers & Operations Research*, 33:3535–3548, 2006.

[4] E. G. Birgin, J. M. Martínez, W. F. Mascarenhas e D. P. Ronconi. Method of Sentinels for Packing Objects within Arbitrary Regions. *Journal of the Operational Research Society*, 57:735–746, 2006.

[5] E. G. Birgin, R. Morabito and F. H. Nishihara. A note on an *L*-approach for solving the manufacturer's pallet loading problem. *Journal of the Operational Research Society*, 56:1448–1451, 2005.

[6] E. Bischoff e W. B. Dowsland. An Application of the Micro to Product Design and Distribution. *Journal of the Operational Research Society*, 33(3):271–280, 1982.

[7] P. De Cani e A. Smith. An Algorithm to Optimize the Layout of Boxes in Pallets. *Journal of the Operational Research Society*, 31(7):573–578, 1980.

[8] K. A. Dowsland. Determining an upper bound for a class of rectangular packing problems. *Computers and Operations Research*, 12(2):201–206, 1985.

[9] K. A. Dowsland. A combined database and algorithmic approach to the pallet loading problem. *Journal of the Operational Research Society*, 38:341–345, 1987.

[10] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[11] T. Hodgson. A combined approach to the pallet loading problem. *IIE Transactions*, 14(3):175–182, 1982.

[12] L. A. N. Lorena e G. M. Ribeiro. Optimizing the woodpulp stowage using Lagrangean relaxation with clusters. Por aparecer em *Journal of the Operational Research Society*, 2006.

[13] L. Lins, S. Lins e R. Morabito. An *L*-approach for packing (ℓ, w)-rectangles into rectangular and *L*-shaped pieces. *Journal of the Operational Research Society*, 54(7):777–789, 2003.

[14] R. Morabito e S. Morales. A simple and effective recursive procedure for the manufacturer's pallet loading problem. *Journal of the Operational Research Society*, 49(8):819–828, 1998.

[15] J. Nelissen. New approaches to the pallet loading problem. Technical report, RWTH Aachen, 1993.

[16] G. Scheithauer e J. Terno. The G4-Heuristic for the Pallet Loading Problem. *Journal of the Operational Research Society*, 47(4):511–522, 1996.

[17] H. J. Steudel. Generating Pallet Loading Patterns: A Special Case of the Two-Dimensional Cutting Stock Problem. *Management Science*, 25(10):997–1004, 1979.