Empacotamento de retângulos

Fábio Henrique Nishihara
Orientador: Ernesto G. Birgin

8 de dezembro de 2003

Resumo

Um dos problemas mais clássicos de empacotamento é encontrar o número máximo de objetos que podem ser empacotados em uma dada região.

Empacotar ítens (por exemplo, caixas) em objetos maiores (palete) e cortar objetos (por exemplo, couro) para produzir ítens menores (carteiras) são problemas similares. O problema de empacotar um dado conjunto de peças em uma dada região maximizando o número total de peças, ocorrem em grande número de situações práticas, incluindo o carregamento de paletes do produtor e o estabelecimento de *layout* em indústrias de roupas. Muitos artigos foram publicados lidando com o problema de empacotamento. Um dos problemas mais populares e úteis nessa área é encontrar o número máximo de retângulos que podem ser empacotados ortogonamente em um retângulo maior.

Neste trabalho estudamos dois enfoques diferentes para resolver o problema de empacotamento de retângulos. Nos dois enfoques consideramos que os retângulos devem ser empacotados ortogonalmente, isto é, seus lados paralelos aos eixos.

Na primeira parte implementamos e testamos o algoritmo L para o empacotamento de retângulos em retângulos, recentemente introduzido em [26]. Este algoritmo é recursivo e empacota retângulos de dimensões (l, w) em peças com formato de L e em peças retangulares maiores. O algoritmo particiona recursivamente um retângulo ou um L em duas novas peças, cada uma podendo ser de novo um retângulo ou um L.

Na segunda parte introduzimos modelos contínuos para o empacotamento de retângulos em conjuntos convexos arbitrários. Esses modelos estão baseados em [6] e em [7]. Experimentos numéricos mostram que tanto os modelos introduzidos como a metodologia proposta para resolver os problemas de otimização contínua são promissores.

Palavras-chaves: problemas de empacotamento, problemas de corte, carregamento de paletes, algoritmo recursivo, modelos contínuos, sentinelas

Sumário

1 Introdução

Empacotar ítens (por exemplo, caixas) em objetos maiores (palete) e cortar objetos (por exemplo, couro) para produzir ítens menores (carteiras) são problemas similares. O problema de empacotar um dado conjunto de peças em uma dada região maximizando o número total de peças, ocorrem em grande número de situações práticas, incluindo o carregamento de paletes do produtor e o estabelecimento de layout em indústrias de roupas. Muitos artigos foram publicados lidando com o problema de empacotamento. Um dos problemas mais populares e úteis nessa área é encontrar o número máximo de retângulos que podem ser empacotados ortogonamente em um retângulo maior.

O problema de carregamento de paletes pode ser visto como um problema de corte e empacotamento bidimensional [16], que consiste em determinar como arranjar retângulos (face das caixas) dentro de um retângulo maior (superfície do palete), de maneira a maximizar a utilização da superfície do palete. Supõe-se que as caixas, em geral disponíveis em grandes quantidades, devem ser arranjadas ortogonalmente, isto é, com seus lados paralelos aos lados do palete.

Hodgson [22], ao estudar o PCP, dividiu-o em dois possíveis casos: o PCP do produtor e o PCP do distribuidor. No primeiro caso, o produtor fabrica um bem que é embalado em caixas iguais de dimensões (l, w, h). Estas caixas devem ser então carregadas em camadas horizontais sobre paletes de dimensões (L, W, H) (onde H é a altura máxima tolerável do carregamento). Em cada camada, uma orientação vertical das caixas é fixada. No segundo caso, o distribuidor recebe vários bens de diversos fornecedores, que estão embalados em caixas diferentes de dimensões $(l_i, w_i, h_i), i = 1, ..., m$, que, então, serão carregadas sobre paletes. Nesse trabalho vamos trabalhar com caixas de dimensões iguais.

A motivação para estudar o carregamento de paletes, além de ser um problema díficil de otimização combinatória, é que ele é importante nas atividades de armazenagem, movimentação e transporte de produtos. Devido à escala de certos sistemas logísticos, um pequeno aumento do número de produtos carregados sobre cada palete pode resultar numa economia global significativa. Alguns estudos propõem métodos exatos para resolver o problema, tais como Dowsland [14], Tsai et al. [42] e Bharttacharya et al. [3]. Métodos heurísticos podem ser encontrados, por exemplo, em Steudel [40], Smith e De Cani [38], Bischoff e Dowsland [10], Nelissen [33, 34], Arenales e Morabito [29], Scheithauer e Terno [37], Herbert e Dowsland [20], Scheithauer e Sommerweiss [36], Morabito e Morales [31, 32], G e Kang [19] e Lins et al. [25]. Limites superiores para o problema foram estudados em Nelissen [34] , Letchford e Amaral [23] e Morabito e Farago [30]. Outras referências podem ser encontradas em Hodgson [22], Dowsland e Dowsland [15], Sweeney e Paternoster [41], Dyckhoff e Finke [17], Balasubramanian [2], Martello [28], Bischoff e Waescher [11], Dyckhoff et al. [18], Arenales et al. [1], Wang e Waescher [43] e Hifi [21].

Este trabalho apresenta duas abordagens para resolver o problema de empacotamento de retângulos. Uma abordagem apresenta o algoritmo L para empacotamento de retângulos em retângulos e a outra, apresenta modelos contínuos para o empacotamento de retângulos em conjuntos convexos arbitrários.

O trabalho está organizado da seguinte maneira. Na primeira parte do trabalho descrevemos o algoritmo L, a nossa implementação e os resultados obtidos com esse algoritmo. Na segunda parte do trabalho descrevemos os modelos contínuos e os resultados obtidos com esses modelos. Por fim, comentamos algumas conclusões.

2 Algoritmo L

2.1 Introdução

Nessa primeira parte do trabalho apresentamos a aproximação L baseada sobre uma função definida recursivamente para empacotar retângulos (l, w) em peças retangulares maiores ou em formato de L. Essa aproximação resolve instâncias difíceis de carregamento de paletes. Como não foi encontrado nenhum exemplo em que a aproximação L falha, os autores de [26] conjecturam que essa aproximação sempre encontra a solução ótima para o empacotamento de retângulos.

Essa primeira parte do trabalho está organizada da seguinte maneira. Na Seção 2.2 descrevemos o algoritmo L para carregamento de paletes introduzido em [26]. Na Seção 2.3 descrevemos a nossa implementação do algoritmo L. Na Seção 2.4 apresentamos alguns resultados numéricos e a solução de 16 problemas que não foram resolvidos por outras heurísticas. Na Seção 2.5 comentamos algumas conclusões.

2.2 Algoritmo

A idéia central do algoritmo é particionar recursivamente um retângulo ou um L em duas peças, cada uma podendo ser de novo um retângulo ou um L.

Restringimos os parâmetros X, Y, x, y, definidos a seguir, como sendo inteiros. Definimos uma peça $L(X, Y, x, y)$ na *posição padrão*, onde $X \geq x$ e $Y \geq y$, como um retângulo cuja diagonal vai de $(0, 0)$ a (X, Y) menos o retângulo cuja diagonal vai de (x, y) a (X, Y) (como mostrado na Figura 1). Assim todos os L's podem ser representados com apenas uma quádupla. A área de $L(X, Y, x, y)$ é dada por $XY - (X - x)(Y - y)$. Em particular, $L(X, Y, X, Y) = R(X, Y)$ é o retângulo cuja diagonal vai $(0, 0)$ a (X, Y). Note que $R(X, Y)$ pode ser visto com um L em que $x = X$ ou $y = Y$. Um $L(X, Y, x, y)$ que é um retângulo é dito *degenerado*. Um $L(X, Y, x, y)$ é dito *não-degenerado* se $0 < x < X$ e $0 < y < Y$.

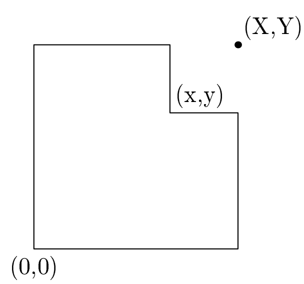


Figura 1: Posição padrão $L(X, Y, x, y)$

Existem cinco distintos modos de subdividir um L não degenerado em dois L's de área menor (como mostrado na Figura 2) e existem dois modos distintos de subdividir um L degenerado (retângulo) em dois L's de área menor (como mostrado na Figura 3).

Dado $L = L(X, Y, x, y)$, denotamos por $P'_k(L)$, com $k \in \{1, 2, 3, 4, 5, 6, 7\}$, o conjunto dos pontos de todas as possíveis subdivisões B_k em L. Os pontos da subdivisão B_k, com $k \in \{1, 2, 3, 4, 5\}$, podem ser representados apenas com uma dupla e os pontos de subdivisão B_6 e B_7, apenas com

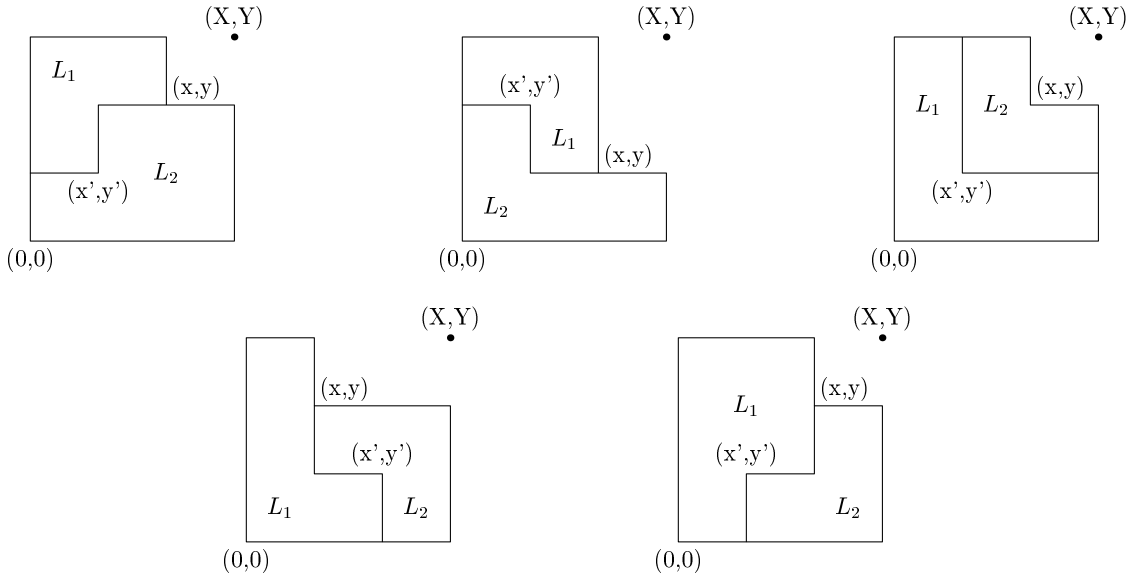


Figura 2: Subdivisões B_1, B_2, B_3, B_4 e B_5 de um L em dois L's.

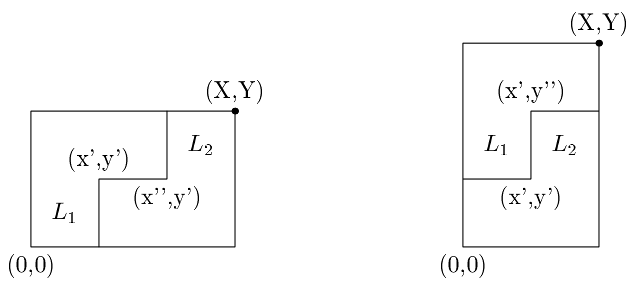


Figura 3: Subdivisões B_6 e B_7 de um retângulo em dois L's.

uma tripla (veja as Figuras 2 e 3). Para cada subdivisão B_k, com $k \in \{1, 2, 3, 4, 5, 6, 7\}$, existe um intervalo de valores válidos para esses pontos e, portanto, cada $P'_k(L)$ pode ser definido como:

$$P'_1(L) = \{p_1 = (x', y') \mid 0 \leq x' \leq x, 0 \leq y' \leq y\}$$
$$P'_2(L) = \{p_2 = (x', y') \mid 0 \leq x' \leq x, y \leq y' \leq Y\}$$
$$P'_3(L) = \{p_3 = (x', y') \mid 0 \leq x' \leq x, 0 \leq y' \leq y\}$$
$$P'_4(L) = \{p_4 = (x', y') \mid x \leq x' \leq X, 0 \leq y' \leq y\}$$
$$P'_5(L) = \{p_5 = (x', y') \mid 0 \leq x' \leq x, 0 \leq y' \leq y\}$$
$$P'_6(L) = \{p_6 = (x', x'', y') \mid 0 \leq x' \leq x'' \leq X, 0 \leq y' \leq Y\}$$
$$P'_7(L) = \{p_7 = (x', y', y'') \mid 0 \leq x' \leq X, 0 \leq y' \leq y'' \leq Y\}$$

Para cada subdivisão B_k, com $k \in \{1,2,3,4,5,6,7\}$, definimos duas funções $\ell_1(L,k,p_k)$ e $\ell_2(L,k,p_k)$ que fazem com que os dois novos L's gerados pela subdivisão B_k fiquem na posição padrão. Essas funções são definidas como:

$$\begin{aligned}
B_1 : \ell_1(L,1,p_1) &= L(x, Y-y', x', Y-y)\\
\ell_2(L,1,p_1) &= L(X, y, X-x', y')\\
B_2 : \ell_1(L,2,p_2) &= L(x, Y-y, x-x', Y-y')\\
\ell_2(L,2,p_2) &= L(X, y', x', y)\\
B_3 : \ell_1(L,3,p_3) &= L(X, Y, x', y')\\
\ell_2(L,3,p_3) &= L(X-x', Y-y', x-x', y-y')\\
B_4 : \ell_1(L,4,p_4) &= L(x', Y, x, y')\\
\ell_2(L,4,p_4) &= L(X-x, y, X-x', y-y')\\
B_5 : \ell_1(L,5,p_5) &= L(x, Y, x', Y-y')\\
\ell_2(L,5,p_5) &= L(X-x', y, X-x, y')\\
B_6 : \ell_1(L,6,p_6) &= L(x'', Y, x', Y-y')\\
\ell_2(L,6,p_6) &= L(X-x', Y, X-x'', y')\\
B_7 : \ell_1(L,7,p_7) &= L(X, Y-y', x', Y-y'')\\
\ell_2(L,7,p_7) &= L(X, y'', X-x', y')
\end{aligned}$$

Quando uma peça é dividida em duas menores, para que a divisão seja uma divisão de fato, as duas peças menores deve ter área maior que 0. Portanto, podemos descartar as subdivisões em que uma das peças é nula. Definimos então $P_k(L)$ como o subconjunto de $P'_k(L)$, tal que $P_k(L) = \{p_k \in P'_k(L) \mid A(\ell_1(L,k,p_k)) > 0, A(\ell_2(L,k,p_k)) > 0\}$, onde $A(L)$ é a área de L, para $k \in \{1,2,3,4,5,6,7\}$.

Dado um L podemos definir limitantes inferiores e superiores para a quantidade de retângulos que podem ser empacotados. Um limite superior $limSup(L)$ pode ser definido como o maior inteiro que é menor que $A(L)/(lw)$. Explicitamente:

$$limSup(L) = \lfloor A(L)/(lw) \rfloor.$$

Um limite inferior $limInf(L)$ pode ser definido como o número máximo de retângulos (l,w) contidos em um empacotamento homogêneo. Empacotamento homogêneo é quando os retângulos são empacotados em apenas uma orientação. Definimos:

- se L é degenerado

$$limInf(L) = \max\{\lfloor X/l \rfloor \lfloor Y/w \rfloor, \lfloor X/w \rfloor \lfloor Y/l \rfloor\},$$

- se L é não degenerado

$$limInf(L) = \max\{limInf(R(x,Y)) + limInf(R(X-x,y)), limInf(R(x,Y-y)) + limInf(R(X,y))\}.$$

Definimos $nRet(L)$ recursivamente como:

- se $limInf(L) = limSup(L)$
$$nRet(L) = limInf(L),$$

- caso contrário
$$nRet(L) = \max_{1\leq k\leq 7}\{\max_{p_k\in P_{k(L)}}\{nRet(\ell_1(L,k,p_k)) + nRet(\ell_2(L,k,p_k))\}\}.$$

A idéia básica é que se o valor de $nRet(L) = n$, então é possível empacotar n retângulos de dimensões (l,w) em L. O algoritmo L consitem em, dado L, calcular a função $nRet(L)$. Para reduzir o custo do cálculo da função $nRet$, vamos reduzir o domínio dessa função.

Seja $Z^{l,w}$ a seqüência infinita $\{0 = z_0 < z_1 < \ldots < z_p < \ldots\}$, de combinações lineares não-negativas de l's e w's. Definimos a grade de pontos de (l,w) como o conjunto dos pontos (z_i, z_j) com $z_i, z_j \in Z^{l,w}$.

Proposição 1 *Um empacotamento ortogonal de $L(X,Y,x,y)$ por retângulos (l,w) pode ser transformado em outro com o mesmo número de retângulos em que cada vértice de cada retângulo está num ponto da grade de (l,w).*

Prova Veja [26].

Com a proposição acima, todas as peças $L(X,Y,x,y)$ que aparecem no cálculo de $nRet$ podem ser trocadas por $L(z_I, z_J, z_i, z_j)$, onde $z_I \leq X \leq z_{I+1}$, $z_J \leq Y \leq z_{J+1}$, $z_i \leq x \leq z_{i+1}$ e $z_j \leq y \leq z_{j+1}$. Para simplificar a notação, denotamos $L(z_I, z_J, z_i, z_j)$ como $L(I,J,i,j)$.

Temos que ajustar a definições das funções $\ell_1(L,k,p_k)$ e $\ell_2(L,k,p_k)$, pois em $Z^{l,w}$ a diferença não é fechada. Portanto, a coordenada $z_r - z_s$ que aparece em $\ell_1(L,k,p_k)$ ou $\ell_2(L,k,p_k)$ deve ser trocada por:

$$z_{r\ominus s} = \max\{z_h \mid z_h \leq z_r - z_s\}$$

Trocamos também algumas notações:

- $L = L(X,Y,x,y)$ por $L = L(I,J,i,j)$,
- $p_k = (x', y')$ por $p_k = (i', j')$, $k = 1,2,3,4,5$,
- $p_6 = (x', x'', y')$ por $p_6 = (i', i'', j')$,
- $p_7 = (x', y', y'')$ por $p_7 = (i', j', j'')$.

Também, podemos diminuir o número de peças a serem consideradas através das eliminações de peças simétricas e de L's degenerados que não são explicitamente retângulos. Duas peças são simétricas, se elas são iguais através de rotações ou reflexões. Um $L(I, J, i, j)$ degenerado não é explicitamente um retângulo, se ($I = i$ e $j < J$) ou ($J = j$ e $i < I$).

Durante a execução do algoritmo, podem aparecer quáduplas (I, J, i, j) que não estão normalizadas. Normalizar significa eliminar peças simétricas (usando apenas peças "deitadas") e eliminar L's degenerados que não são explicitamente retângulos. Essas quáduplas são normalizadas para:

1. se $i = 0$ então $(I, J, i, j)^N = (I, j, I, j)$,
2. se $j = 0$ então $(I, J, i, j)^N = (i, J, i, J)$,
3. se $I = i$ ou $J = j$ então $(I, J, i, j)^N = (I, J, I, J)$,
4. se $I = i$ e $J = j$ e $I < J$ então $(I, J, i, j)^N = (J, I, j, i)$,
5. se $0 < i < I$ e $0 < j < J$ e $I < J$ então $(I, J, i, j)^N = (J, I, j, i)$,
6. se $0 < i < I$ e $0 < j < J$ e $I = J$ e $i < j$ então $(I, J, i, j)^N = (J, I, j, i)$,
7. caso contrário $(I, J, i, j)^N = (I, J, i, j)$.

Os itens $(1) - (3)$ são para eliminar L's degenerados que não são explicitamente retângulos. Os itens $(4) - (6)$ são para deitar as peças. Para ser mais claro, o item (4) serve para deitar retângulos e os itens (5) e (6) são para deitar L's.

A seguir definimos as funções $\ell_1^N(L, k, p_k)$ e $\ell_2^N(L, k, p_k)$ para que a partição de um retângulo ou de um L resulte em retângulos ou L's normalizados:

$$
\begin{aligned}
B_1 : \ell_1^N(L, 1, p_1) &= L((i, J \ominus j', i', J \ominus j)^N) \\
\ell_2^N(L, 1, p_1) &= L((I, j, I \ominus i', j')^N) \\
B_2 : \ell_1^N(L, 2, p_2) &= L((i, J \ominus j, i \ominus i', J \ominus j')^N) \\
\ell_2^N(L, 2, p_2) &= L((I, j', i', j)^N) \\
B_3 : \ell_1^N(L, 3, p_3) &= L((I, J, i', j')^N) \\
\ell_2^N(L, 3, p_3) &= L((I \ominus i', J \ominus j', i \ominus i', j \ominus j')^N) \\
B_4 : \ell_1^N(L, 4, p_4) &= L((i', J, i, j')^N) \\
\ell_2^N(L, 4, p_4) &= L((I \ominus i, j, I \ominus i', j \ominus j')^N) \\
B_5 : \ell_1^N(L, 5, p_5) &= L((i, J, i', J \ominus j')^N) \\
\ell_2^N(L, 5, p_5) &= L((I \ominus i', j, I \ominus i, j')^N) \\
B_6 : \ell_1^N(L, 6, p_6) &= L((i'', J, i', J \ominus j')^N) \\
\ell_2^N(L, 6, p_6) &= L((I \ominus i', J, I \ominus i'', j')^N) \\
B_7 : \ell_1^N(L, 7, p_7) &= L((I, J \ominus j', i', J \ominus j'')^N)
\end{aligned}
$$

$$\ell_2^N(L, 7, p_7) = L((I, j'', I \ominus i', j')^N)$$

O *AlgoritmoL* é estruturalmente simples. Ele usa uma função indicadora *solucao* que indica qual tipo de solução foi usado para resolver cada subproblema (indica se foi resolvido com empacotamento homogêneo ou com uma das subdivisões - B1,...,B7). Essa função é inicializada como "não tendo solução". Assim que os valores corretos de *nRet* para cada subproblema são encontrados, *solucao* indica qual tipo de solução foi usado. *ptoDiv* guarda o ponto de subdivisão da solução usada. Se o tipo de solução para uma peça é o empacotamento homogêneo, então *ptoDiv* fica indefinido para ela.

A seguir descrevemos a função *AlgoritmoL*.

```
AlgoritmoL(L) {
  Aloca espaço para armazenar:
  - nRet(L)
  - solucao(L)
  - ptoDiv(L)
  Inicializa solucao(L) como não tendo solução.
  Resolve(L)
  DesenhaSolucao(L)
}
```

O *AlgoritmoL* chama a função recursiva *Resolve*. Para cada L, se $limInf(L) = limSup(L)$ o valor de *solucao* para essa peça L é o empacotamento homogêneo, que é fácil de calcular e não precisa mais ser subdividida. O valor de *ptoDiv* para esse L fica indefinido. Caso contrário, *Resolve* examina recursivamente cada subdivisão B_k (isto é, $\ell_1(L, k, p_k)$ e $\ell_2(L, k, p_k)$) para todos os pontos em P_k.

A seguir descrevemos a função recursiva *Resolve*.

```
Resolve(L) {
  se solucao(L) não tem solução então
    nRet(L) ← limInf(L)
    solucao(L) ← empacotamento homogêneo
    se limInf(L) ≠ limSup(L) então
      para k ∈ {B1, ..., B7} faça
        para pk ∈ ptoDiv(L) faça
          L1 ← ℓ1(L, k, pk)
          L2 ← ℓ2(L, k, pk)
          se limSup(L1) + limSup(L2) > nRet(L) então
            Resolve(L1)
            Resolve(L2)
            se nRet(L1) + nRet(L2) > nRet(L) então
              nRet(L) ← nRet(L1) + nRet(L2)
              solucao(L) ← k
              ptoDiv(L) ← pk
}
```

Depois de encontrar a melhor solução para o L, o *AlgoritmoL* chama a função recursiva *DesenhaSolucao* para desenhar a solução do problema. Desenhar significa descobrir as coordenadas dos retângulos em relação ao retângulo maior.

A função *DesenhaSolucao* é recursiva. Se o tipo de solução é o empacotamento homogêneo, então são desenhados os retângulos. Caso contrário, *DesenhaSolucao* divide a peça, usando subdivisão indicada em *solucao*, e chama recursivamente para cada uma das novas peças geradas.

No final da função *Resolve*, a solução para o problema não está de uma forma muito amigável. Todas as peças estão na posição padrão e normalizadas. Portanto, as coordenadas dos retângulos menores, que estão dentro da peça, estarão em relação a essa peça e não ao retângulo maior. A função recursiva *DesenhaSolucao* verifica qual solução foi usada e coloca as coordenadas dos retângulos em relação ao retângulo maior. No final dessa função, todos as coordenadas dos retângulos estarão em relação ao retângulo maior.

Seja $R(x_1, y_1, x_2, y_2)$, onde (x_1, y_1) é o vértice inferior esquerdo do retângulo e (x_2, y_2) é o vértice superior direito do retângulo (como mostrado na Figura 4). Para representar um retângulo é necessário de apenas uma quádupla. Com essa quádupla é possível descobrir as coordenadas dos quatro vértices do retângulo.

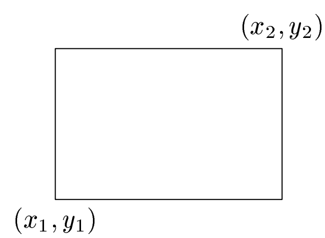


Figura 4: Representação de um retângulo

Dado um $L(X, Y, x, y)$ na posição padrão e normalizado, queremos descobrir como essa peça estava antes. Denotamos por T_k, com $k \in \{1, ..., 8\}$, as transformações que são necessárias para deixar essa peça como estava antes de ser colocada na posição padrão e de ser normalizada. Antes, essa peça L poderia estar em oito posições (como mostrado na Figura 5).

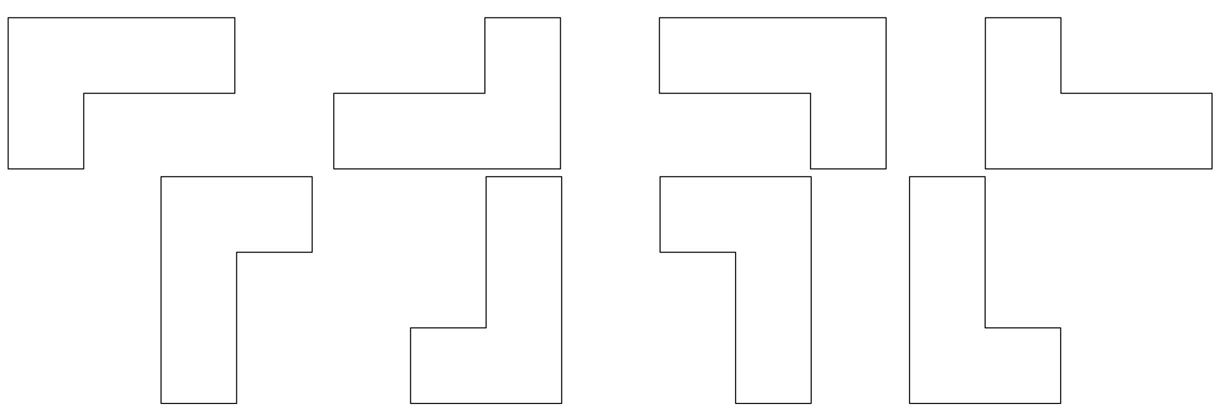


Figura 5: Posições que uma peça poderia estar.

Cada uma dessas tranformações T_k, com $k \in \{1, ..., 8\}$ é definida como:

$$T_1(x_0, y_0) = (x_0, Y - y_0)$$
$$T_2(x_0, y_0) = (X - x_0, y_0)$$
$$T_3(x_0, y_0) = (X - x_0, Y - y_0)$$
$$T_4(x_0, y_0) = (x_0, y_0)$$
$$T_5(x_0, y_0) = (y_0, X - x_0)$$
$$T_6(x_0, y_0) = (Y - y_0, x_0)$$
$$T_7(x_0, y_0) = (Y - y_0, X - x_0)$$
$$T_8(x_0, y_0) = (y_0, x_0)$$

Quando uma peça é dividida, ela gera duas novas peças, L_1 e L_2. Para cada peça gerada de cada subdivisão, pode ser possível usar duas transformações, mas apenas uma será usada. Na Tabela 1 são mostradas as tranformações possíveis para cada um. Por exemplo, usando a subdivisão B_1, a peça L_1, que está na posição padrão e normalizada, pode usar a transformação T_1 ou T_5 (mas usará apenas uma delas) para voltar à posição que estava antes. Para descobrir qual das duas transformações usar, devemos verificar a largura e a altura de cada peça depois de colocá-las na posição padrão, sem normalizá-las, eliminado apenas os L's degenerados que não são explicitamente retângulos.

Subdivisão	L_1	L_2
B_1	T_1 ou T_5	T_2 ou T_6
B_2	T_3 ou T_7	T_4 ou T_8
B_3	T_4	T_8
B_4	T_4 ou T_8	T_3 ou T_7
B_5	T_1 ou T_5	T_2 ou T_6
B_6	T_1 ou T_5	T_2 ou T_6
B_7	T_1 ou T_5	T_2 ou T_6

Tabela 1: Tranformações para cada subdivisão.

Em particular, se $L(X, Y, X, Y) = R(X, Y)$ então usamos as tranformações T_4 ou T_8.

Depois dessa transformação precisamos transladar a peça para que fique na sua posição correta.

A seguir descrevemos a função recursiva *DesenhaSolucao*.

```
DesenhaSolucao(L) {
  se solucao(L) = empacotamento homogêneo então
    desenha os retângulos
  senão
    dividir a peça usando solucao(L) no ponto ptoDiv(L)
    colocar as duas novas peças (L_1, L_2) na posição padrão
    DesenhaSolucao(L_1^N) // L_1 normalizada
    usar a tranformação adequada para essa peça
    transladar a peça
    DesenhaSolucao(L_2^N) // L_2 normalizada
    usar a tranformação adequada para essa peça
    transladar a peça
}
```

2.3 Implementação

A maior dificuldade do método está na quantidade de memória que o algoritmo necessita. Precisa-se armazenar, para cada subproblema, a quantidade de retângulos empacotados, o tipo de solução utilizado e o ponto de divisão (informação complementar ao tipo de solução). Em [26], esses dados foram armazendos usando uma estrutura de dados chamado PHORMA (Perfect Hashed Order Restricted Multidimensional Array) [24, 12]. Essa estrutura de dados usa dígrafos para indexar as peças.

Seja p e q as cardinalidades dos conjuntos $c \leq L | c = \alpha l + \beta w, com \alpha, \beta \in I\!N$ e $c \leq W | c = \alpha l + \beta w, com \alpha, \beta \in I\!N$ respectivamente. Como, por definição, $L \geq W$, então $p \geq q$. Na nossa implementação armazenamos esse dados em 3 vetores de tamanho p^4 cada um.

Para ter acesso direto aos dados precisamos, dado um subproblema $L(I, J, i, j)$, calcular o índice dos vetores onde seus dados estão armazenados. Para isto, fizemos uma função bijetora $\alpha : I\!N \mapsto I\!N^4$ que serve para associar cada $L(I, J, i, j)$ a um índice k. Seja m o tamanho da largura do retângulo maior. Então definimos essa função bijetora como:

$$\alpha(k) = ((k/m^3) \bmod m, (k/m^2) \bmod m, (k/m) \bmod m, k \bmod m),$$

$$\alpha^{-1}(I, J, i, j) = (((I \times m) + J) \times m + i) \times m + j,$$

onde $a \bmod b$ é o resto da divisão de a por b.

Usando PHORMA, reduz-se a quantidade de memória necessária para resolver os problemas, mas em compensação demora mais comparando com a nossa implementação. Com a nossa implementação, tivemos alguns problemas devido à grande quantidade de memória que o algoritmo necessitava para resolver alguns problemas. Note que não são usadas todas as posições dos vetores que armazenam os dados (veja Tabela 3 na Seção 2.4).

2.4 Experimentos Numéricos

Nessa seção aplicamos a nossa implementação do algoritmo L nos dezesseis problemas apresentados em [26]. Esses problemas são considerados problemas difíceis de resolver.

A nossa implementação do algoritmo L foi feita em C. Abaixo estão as configurações das três máquinas usadas para resolver os problemas:

- xingu: AMD Athlon 1.1 Gb, 256 Mb de memória.
- jacuzzi: Pentium 3 600 MHz, 512 Mb de memória.
- sutra: AMD Athlon 1.5 Gb, 3.5 Gb de memória.

As Figuras (6)-(21) ilustram as soluções obtidas pelo algoritmo.

Na Tabela 2 apresentamos, para cada problema, o tempo gasto para resolver o problema em cada uma das máquinas descritas acima (tempo gasto com a função *Resolve*, portanto sem contar o tempo gasto para as inicializações, alocamento de memória e a função *DesenhaSolucao*).

Observe que $P_n = (L, W, l, w)$ significa que dado um retângulo de dimensão (L, W) conseguimos empacotar n retângulos de dimensões (l, w).

	xingu	jacuzzi	sutra
$P_{53} = (43, 26, 7, 3)$	4,867	7,867	3,583
$P_{57} = (49, 28, 8, 3)$	14,450	23,617	10,783
$P'_{69} = (57, 34, 7, 4)$	26,067	42,400	19,433
$P''_{69} = (63, 44, 8, 5)$	29,567	48,200	22,033
$P_{71} = (61, 35, 10, 3)$	68,683	110,250	51,283
$P_{75} = (67, 37, 11, 3)$	120,683	190,700	89,667
$P'_{77} = (61, 38, 10, 3)$	104,783	167,650	78,033
$P''_{77} = (61, 38, 6, 5)$	36,367	59,217	27,167
$P_{81} = (67, 40, 11, 3)$	181,433	287,267	136,250
$P'_{82} = (74, 49, 11, 4)$	268,767	425,817	201,633
$P''_{82} = (93, 46, 13, 4)$	-	504,033	238,717
$P'_{96} = (106, 59, 13, 5)$	-	1023,320	482,033
$P''_{96} = (141, 71, 13, 8)$	-	-	203,533
$P_{97} = (74, 46, 7, 5)$	112,783	179,900	84,017
$P_{99} = (86, 52, 9, 5)$	387,250	612,500	289,717
$P_{100} = (108, 65, 10, 7)$	-	346,100	159,117

Tabela 2: Tempo (em segundos) gasto para resolver os dezesseis problemas difíceis.

Em algumas máquinas nem todos os problemas foram resolvidos, pois a memória da máquina não foi suficiente.

Na Tabela 3 apresentamos a quantidade de posições alocadas pelo vetor *solucao* e quanto foi usado dele. Note que nesses 16 problemas, não usamos mais que 3% do vetor.

	tamanho do vetor	posições usadas	porcentagem usada do vetor
$P_{53} = (43, 26, 7, 3)$	2085136	20457	0,981%
$P_{57} = (49, 28, 8, 3)$	3418801	47008	1,375%
$P'_{69} = (57, 34, 7, 4)$	5764801	71614	1,242%
$P''_{69} = (63, 44, 8, 5)$	6250000	75923	1,215%
$P_{71} = (61, 35, 10, 3)$	7890481	143698	1,821%
$P_{75} = (67, 37, 11, 3)$	11316496	210270	1,858%
$P'_{77} = (61, 38, 10, 3)$	7890481	188729	2,392%
$P''_{77} = (61, 38, 6, 5)$	7311616	90449	1,237%
$P_{81} = (67, 40, 11, 3)$	11316496	275432	2,434%
$P'_{82} = (74, 49, 11, 4)$	12960000	356428	2,750%
$P''_{82} = (93, 46, 13, 4)$	33362176	433080	1,298%
$P'_{96} = (106, 59, 13, 5)$	47458321	694109	1,463%
$P''_{96} = (141, 71, 13, 8)$	100000000	395713	0,396%
$P_{97} = (74, 46, 7, 5)$	15752961	188579	1,197%
$P_{99} = (86, 52, 9, 5)$	25411681	470589	1,852%
$P_{100} = (108, 65, 10, 7)$	45212176	331070	0,732%

Tabela 3: Uso do vetor alocado para resolver os dezesseis problemas difíceis.

2.5 Conclusões

Nesta primeira parte do trabalho estudamos o algoritmo L para o empacotamento de retângulos introduzido em [26]. A descrição bibliográfica, a descrição do problema e a descrição do algoritmo estão fortemente baseados em [26]. Já na implementação do método utilizamos uma estrutura de dados diferente.

A estrutura de dados utilizada em [26] visa utilizar pouca memória em detrimento ao tempo de acesso aos dados. E, no mesmo trabalho, o tempo de execução é apontado como a maior deficiência do método. Em função disso, na nossa implementação tentamos, utilizando mais memória, diminuir o tempo de execução do algoritmo.

Mais ainda, só uma pequena fração da memória alocada é de fato utilizada. Isso aponta a possibilidade de pesquisar estruturas de dados altenativas que, utilizando menos memória, conservem os ganhos de tempo de execução obtidas na presente implementação.

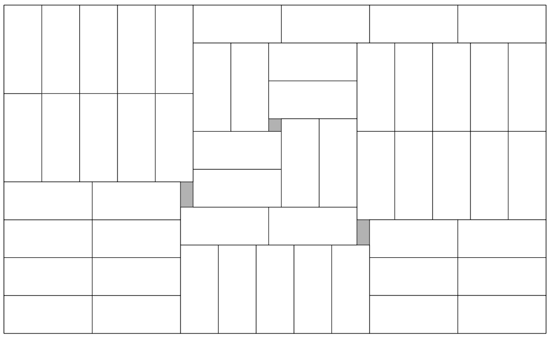


Figura 6: $P_{53} = (43, 26, 7, 3)$

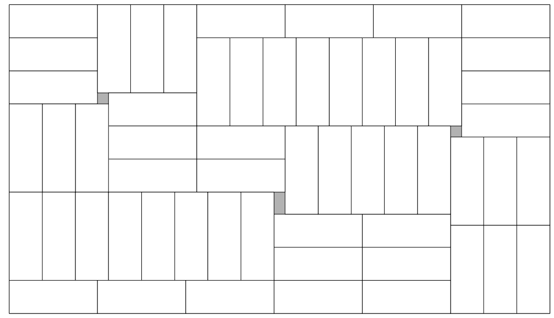


Figura 7: $P_{57} = (49, 28, 8, 3)$

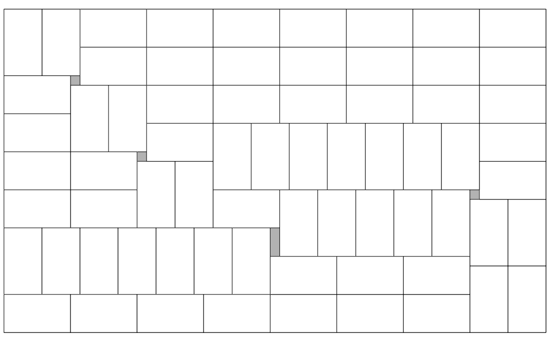


Figura 8: $P'_{69} = (57, 34, 7, 4)$

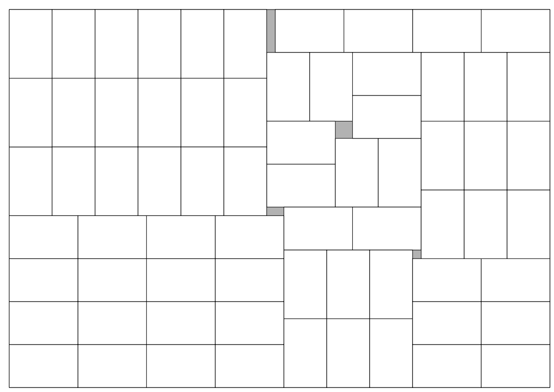


Figura 9: $P''_{69} = (63, 44, 8, 5)$

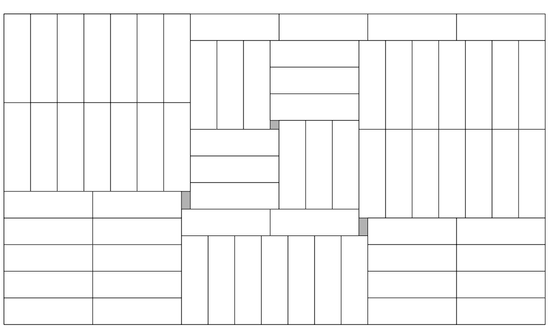


Figura 10: $P_{71} = (61, 35, 10, 3)$

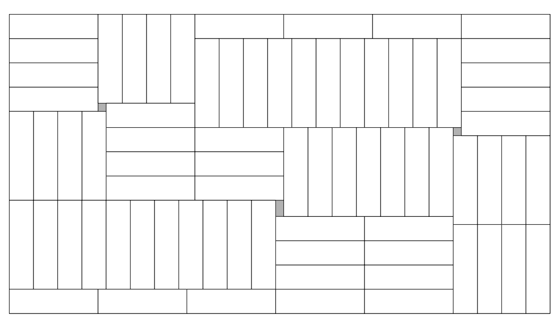


Figura 11: $P_{75} = (67, 37, 11, 3)$

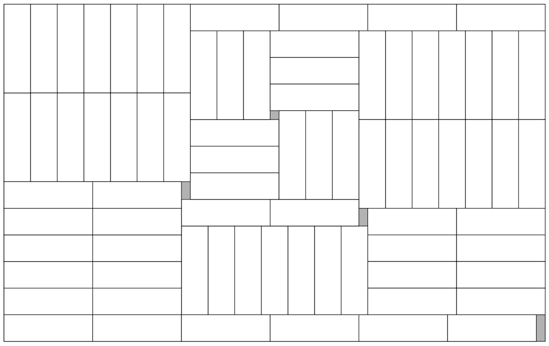


Figura 12: $P'_{77} = (61, 38, 10, 3)$

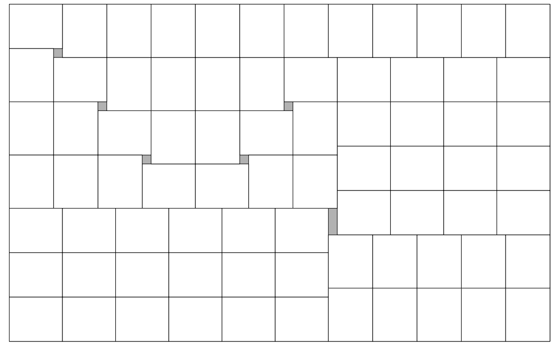


Figura 13: $P''_{77} = (61, 38, 6, 5)$

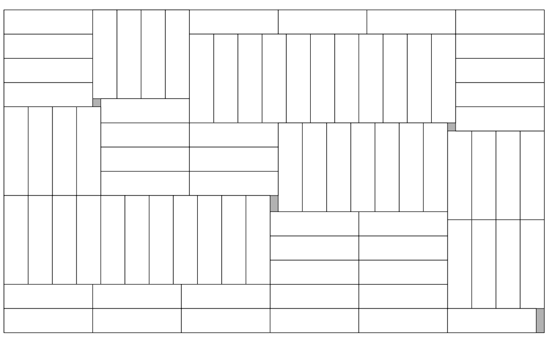


Figura 14: $P_{81} = (67, 40, 11, 3)$

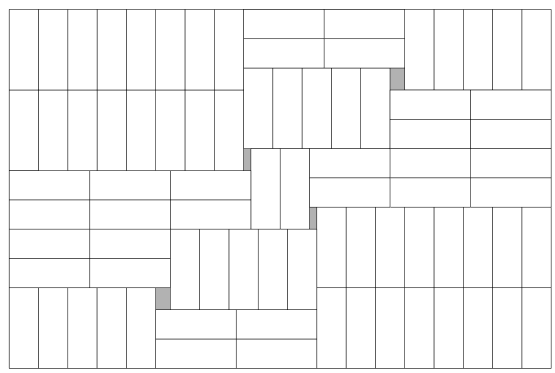


Figura 15: $P'_{82} = (74, 49, 11, 4)$

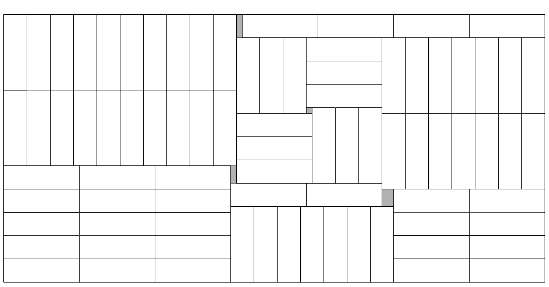


Figura 16: $P''_{82} = (93, 46, 13, 4)$

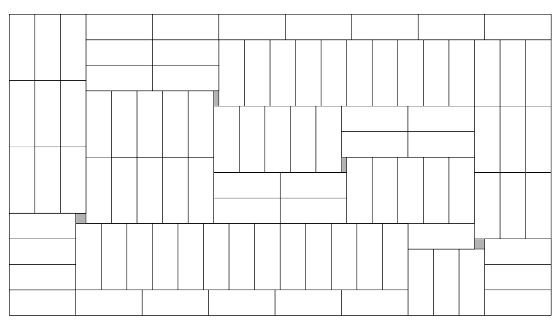


Figura 17: $P'_{96} = (106, 59, 13, 5)$

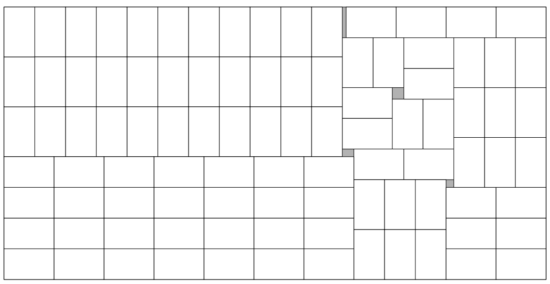


Figura 18: $P''_{96} = (141, 71, 13, 8)$

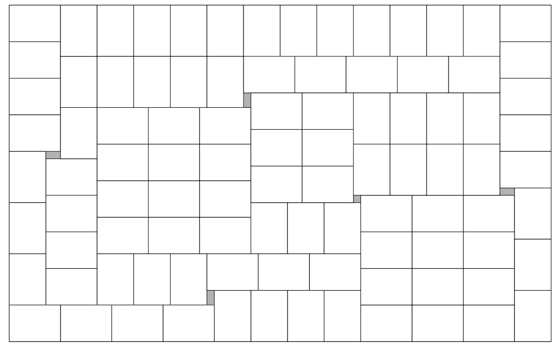


Figura 19: $P_{97} = (74, 46, 7, 5)$

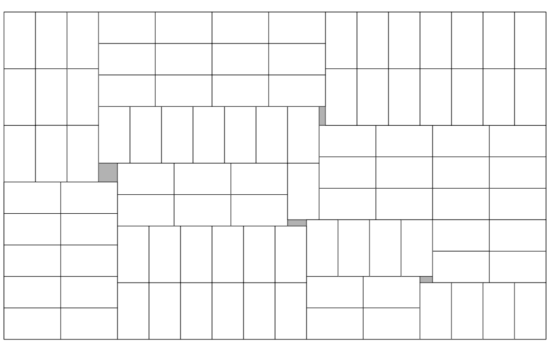


Figura 20: $P_{99} = (86, 52, 9, 5)$

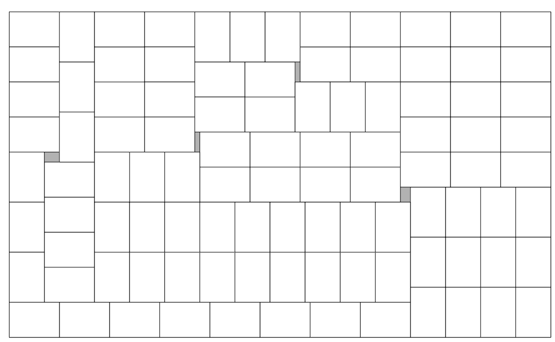


Figura 21: $P_{100} = (108, 65, 10, 7)$

3 Modelos Contínuos

3.1 Introdução

Nessa segunda parte do trabalho abordaremos o problema de empacotamento de retângulos utilizando modelos de otimização contínua. Alguns dos modelos apresentados estão baseados no conceito de *sentinelas* recentemente introduzido em [6].

Basicamente os modelos tentam, de alguma forma, colocar os retângulos dentro de uma região convexa, minimizando a superposição entre eles.

Definimos uma função contínua e diferenciável para cada modelo que mede a superposição entre os retângulos. Essa função vale zero, quando não há superposição entre os retângulos e vale algum valor positivo, caso contrário. Dada essa função tentaremos encontrar um minimizador global.

Apresentaremos um procedimento baseado sobre um problema de decisão não-linear para resolver o problema. Essa formulação é baseada sobre algumas restrições. O procedimento computacional tenta encontrar o número máximo de retângulos que satisfazem essas restrições.

Essa segunda parte do trabalho está organizada da seguinte maneira. Na Seção 3.2 descrevemos o problema de decisão e alguns modelos contínuos para resolver esse problema. Na Seção 3.3 descrevemos um procedimento para empacotar o número máximo de retângulos possíveis. Na Seção 3.4 apresentamos alguns resultados numéricos e a solução dos problemas de empacotamento de retângulos em regiões convexas. Na Seção 3.5 comentamos algumas conclusões.

3.2 Problema de decisão

O problema de decisão tratado nessa seção é:

> *Dados k retângulos de dimensões (l, w), sendo d deitados ($0 \leq d \leq k$) e uma região convexa, verificar se é possível colocar todos os retângulos dentro dessa região ou não.*

Introduziremos alguns modelos para esse problema. Encontrar a resposta para o problema de decisão dependerá de encontrar o minimizador global de um problema de otimização não-linear e não-convexo. Também descreveremos um método para encontrar pontos estacionários de primeira ordem (minimizadores locais) do modelo introduzido e a estratégia para aumentar a probabilidade de encontrar minimizadores globais.

3.2.1 Modelos Contínuos

Antes de mostrar os modelos, vamos definir o que são sentinelas, pois em alguns métodos são necessários o uso deles.

Um conjunto de sentinelas é um conjunto mínimo de pontos de um retângulo, tal que se dois retângulos R_1 e R_2 se intersectam, então ocorre pelo menos uma das seguintes situações:

- pelo menos um sentinela do retângulo R_1 está no interior do retângulo R_2,
- pelo menos um sentinela do retângulo R_2 está no interior do retângulo R_1.

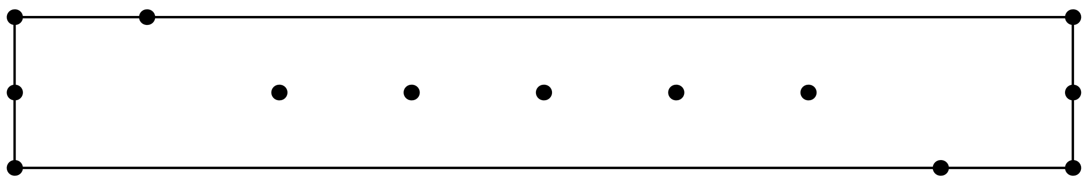


Figura 22: Conjunto de sentinelas de um retângulo

A Figura 22 é um exemplo de um conjunto de sentinelas de um retângulo.

Agora que já definimos o que são sentinelas, vamos mostrar os modelos contínuos.

Método do Vetor

Esse método usa sentinelas. Para verificar se ocorre intersecção entre dois retângulos R_1 e R_2, devemos verificar para todos os sentinelas do retângulo R_1, se existe ao menos um deles no interior do retângulo R_2 e verificar para todos os sentinelas do retângulo R_2, se existe ao menos um deles no interior do retângulo R_1.

Para aplicar o método devemos, antes de verificar se um sentinela de um retângulo está no interior do outro retângulo, levar o retângulo para a origem. Seja (r_x, r_y) a coordenada do vértice inferior esquerdo do retângulo e (s_x, s_y) a coordenada do sentinela. Para levar o retângulo para a origem é só subtrair sua coordenada pelo vetor (r_x, r_y). Também devemos subtrair a coordenada do sentinela por esse vetor. Note que a coordenada do retângulo, depois de subtraído pelo vetor, será sempre $(0, 0)$ (origem), portanto não é necessário subtrair o vetor da coordenada do retângulo, apenas será necessária subtrair o vetor da coordenada do sentinela, assim a nova coordenada do sentinela será $(s_x - r_x, s_y - r_y)$.

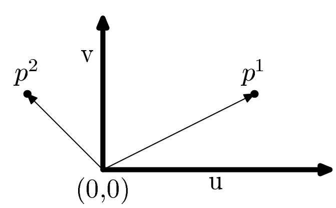


Figura 23: Método do vetor

Após feito isso, podemos considerar que o retângulo, de dimensão (l, w), tem dois vetores u e v, onde $u = (d_x, 0)$ e $v = (0, d_y)$, e se o retângulo R_1 está deitado então $d_x = l$ e $d_y = w$, e se o retângulo R_1 está em pé então $d_x = w$ e $d_y = l$ (como mostra a Figura 23). Assim a nova coordenada $p = (p_x, p_y)$ do sentinela pode ser representada através da seguinte equação linear:

$$p = \alpha \times u + \beta \times v,$$

onde $\alpha, \beta \in I\!R$.

Se $0 < \alpha < 1$ e $0 < \beta < 1$ então o sentinela está no interior do retângulo. Portanto os retângulos se intersectam.

Resolvendo a equação temos que:

$$\alpha = \frac{p_x}{d_x} = \frac{s_x - r_x}{d_x}$$
$$\beta = \frac{p_y}{d_y} = \frac{s_y - r_y}{d_y}$$

A função contínua e diferenciável que mede a superposição entre os retângulos e seu gradiente para esse método são:

$$\begin{aligned}
f(r) &= \sum_{i=1}^{nRet} \sum_{j=1, j \neq i}^{nRet} \sum_{m=1}^{nSent} \max(0,\alpha)^2 \times \max(0,1-\alpha)^2 \times \max(0,\beta)^2 \times \max(0,1-\beta)^2 \\
&= \sum_{i=1}^{nRet} \sum_{j=1, j \neq i}^{nRet} \sum_{m=1}^{nSent} \max(0, \tfrac{s_x^{jk} - r_x^i}{d_x^i})^2 \times \max(0, 1 - \tfrac{s_x^{jk} - r_x^i}{d_x^i})^2 \times \\
&\qquad \max(0, \tfrac{s_y^{jk} - r_y^i}{d_y^i})^2 \times \max(0, 1 - \tfrac{s_y^{jk} - r_y^i}{d_y^i})^2
\end{aligned}$$

$$\nabla f(r) = \begin{pmatrix} \sum_{i=1}^{nRet} \sum_{j=1, j \neq i}^{nRet} \sum_{m=1}^{nSent} \frac{2\alpha(1-\alpha)(2\alpha-1)\beta^2(1-\beta)^2}{d_x^i} \\ \sum_{i=1}^{nRet} \sum_{j=1, j \neq i}^{nRet} \sum_{m=1}^{nSent} \frac{2\beta(1-\beta)(2\beta-1)\alpha^2(1-\alpha)^2}{d_y^i} \\ \vdots \\ \vdots \end{pmatrix}$$

onde $r = (r_x^1, r_y^1, r_x^2, r_y^2, \ldots, r_x^n, r_y^n)$ e r_x^i é a coordenada inferior esquerda x do retângulo i e r_y^i é a coordenada inferior esquerda y do retângulo i. s_x^{jk} é a coordenada x do sentinela k do retângulo j e s_y^{jk} é a coordenada y do sentinela k do retângulo j. E $nRet$ é o número de retângulos e $nSent$ é o número de sentinelas.

Essa função vale *zero* se não existe superposição entre os retângulos e qualquer valor positivo se existe superposição entre os retângulos.

Método das Restrições

Esse método usa sentinelas. Para verificar se ocorre intersecção entre dois retângulos R_1 e R_2, devemos verificar para todos os sentinelas do retângulo R_1, se existe ao menos um deles no interior do retângulo R_2 e verificar para todos os sentinelas do retângulo R_2, se existe ao menos um deles no interior do retângulo R_1.

Podemos considerar que os lados do retângulo R_1, de dimensão (l, w), são quatro restrições lineares (como mostra a Figura 24). Sejam (s_x, s_y) a coordenada de um sentinela do retângulo R_2 e d_x e d_y as dimensões do retângulo R_1, onde se o retângulo está deitado então $d_x = l$ e $d_y = w$, e se o retângulo está em pé então $d_x = w$ e $d_y = l$.

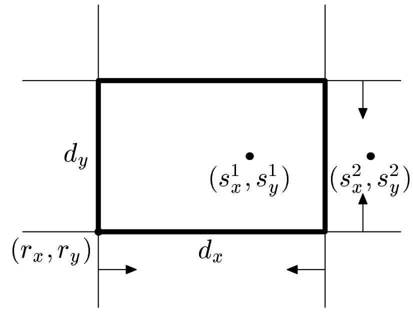


Figura 24: Método das restrições

Assim se:

$$r_x < s_x < r_x + d_x$$

$$e$$

$$r_y < s_y < r_y + d_y$$

então o sentinela está no interior do retângulo. Portanto os retângulos se intersectam.

A função contínua e diferenciável que mede a superposição entre os retângulos e seu gradiente para esse método são:

$$f(r) = \sum_{i=1}^{nRet} \sum_{j=1, j \neq i}^{nRet} \sum_{m=1}^{nSent} \max(0, s_x^{jk} - r_x^i)^2 \times \max(0, r_x^i + d_x^i - s_x^{jk})^2 \times \max(0, s_y^{jk} - r_y^i)^2 \times \max(0, r_y^i + d_y^i - s_y^{jk})^2$$

$$\nabla f(r) = \begin{pmatrix} \sum_{i=1}^{nRet} \sum_{j=1, j \neq i}^{nRet} \sum_{m=1}^{nSent} & 2(s_x^{jk} - r_x^i)(r_x^i + d_x^i - s_x^{jk})((r_x^i + d_x^i - s_x^{jk}) - (s_x^{jk} - r_x^i)) \\ & ((s_y^{jk} - r_y^i)^2 (r_y^i + d_y^i - s_y^{jk})^2) \\ \sum_{i=1}^{nRet} \sum_{j=1, j \neq i}^{nRet} \sum_{m=1}^{nSent} & 2(s_y^{jk} - r_y^i)(r_y^i + d_y^i - s_y^{jk})((r_y^i + d_y^i - s_y^{jk}) - (s_y^{jk} - r_y^i)) \\ & ((s_x^{jk} - r_x^i)^2 (r_x^i + d_x^i - s_x^{jk})^2) \\ \vdots & \\ \vdots & \end{pmatrix}$$

onde $r = (r_x^1, r_y^1, r_x^2, r_y^2, \ldots, r_x^n, r_y^n)$ e r_x^i é a coordenada inferior esquerda x do retângulo i e r_y^i é a coordenada inferior esquerda y do retângulo i. s_x^{jk} é a coordenada x do sentinela k do retângulo j e s_y^{jk} é a coordenada y do sentinela k do retângulo j. E $nRet$ é o número de retângulos e $nSent$ é o número de sentinelas.

Essa função vale *zero* se não existe superposição entre os retângulos e qualquer valor positivo se existe superposição entre os retângulos.

Método da Distância do Centros

Esse método NÃO usa sentinelas.

Considere dois retângulos R_1 e R_2 de dimensões (l, w), onde as coordenadas do centro do retângulo são (r_x^1, r_y^1) e (r_x^2, r_y^2) respectivamente. Sejam d_x^1 e d_y^1 as dimensões do retângulo R_1 e d_x^2 e d_y^2 as dimensões do retângulo R_2, onde se o retângulo está deitado então $d_x = l$ e $d_y = w$, e se o retângulo está em pé então $d_x = w$ e $d_y = l$ (Veja Figura 25).

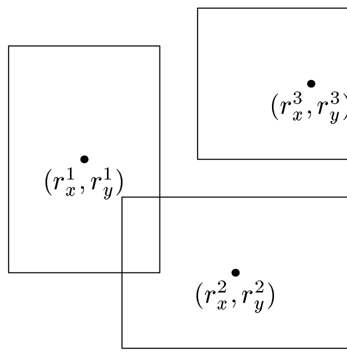


Figura 25: Método da distância dos centros

Assim se:

$$|r_x^1 - r_x^2| < d_x^1 + d_x^2$$
$$e$$
$$|r_y^1 - r_y^2| < d_y^1 + d_y^2$$

então os retângulos se intersectam. Caso contrário, os retângulos não se intersectam.

A função contínua e diferenciável que mede a superposição entre os retângulos e seu gradiente para esse método são:

$$f(r) = \sum_{i=1}^{nRet} \sum_{j=i+1}^{nRet} \max(0, (\frac{d_x^i}{2} + \frac{d_x^j}{2})^2 - (r_x^i - r_x^j)^2)^2 \times \max(0, (\frac{d_y^i}{2} + \frac{d_y^j}{2})^2 - (r_y^i - r_y^j)^2)^2$$

$$\nabla f(r) = \begin{pmatrix} \sum_{i=1}^{nRet} \sum_{j=i+1}^{nRet} -4((\frac{d_x^i}{2} + \frac{d_x^j}{2})^2 - (r_x^i - r_x^j)^2)((\frac{d_y^i}{2} + \frac{d_y^j}{2})^2 - (r_y^i - r_y^j)^2)^2 (r_x^i - r_x^j) \\ \sum_{i=1}^{nRet} \sum_{j=i+1}^{nRet} -4((\frac{d_x^i}{2} + \frac{d_x^j}{2})^2 - (r_x^i - r_x^j)^2)^2 ((\frac{d_y^i}{2} + \frac{d_y^j}{2})^2 - (r_y^i - r_y^j)^2)(r_y^i - r_y^j) \\ \vdots \\ \vdots \end{pmatrix}$$

onde $r = (r_x^1, r_y^1, r_x^2, r_y^2, \ldots, r_x^n, r_y^n)$ e r_x^i é a coordenada x do centro do retângulo i e r_y^i é a coordenada y do dentro do retângulo i. E $nRet$ é o número de retângulos.

Essa função vale *zero* se não existe superposição entre os retângulos e qualquer valor positivo se existe superposição entre os retângulos.

Método da Circunferência

Esse método serve para gerar uma configuração inicial dos retângulos para os outros métodos. Pegamos algumas circunferências que cobre quase todo o retângulo, como mostra a Figura 26. Esse método é citado em [7] para o empacotamento de cilindros.

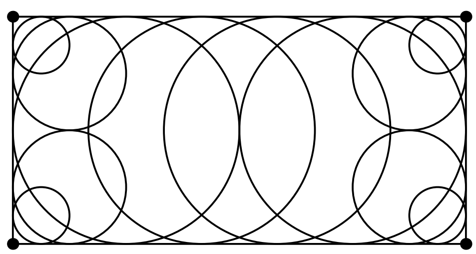


Figura 26: O retângulo coberto parciamente por circunferêcias para gerar configurações iniciais.

Para gerar essa configuração inicial, devemos verificar para todas as circunferências de um retângulo se não existe intersecção com as outras circunferências dos outros retângulos.

Considere duas circunferências C_1 e C_2, onde as coordenadas do centro são (c_x^1, c_y^1) e (r_x^2, r_y^2) e os raios são R^1 e R^2, respectivamente. (Veja Figura 27)

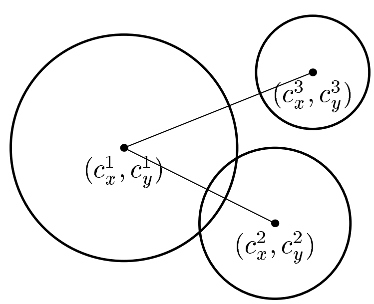


Figura 27: Método da circunferência

Assim se:

$$(r_x^1 - r_x^2)^2 + (r_y^1 - r_y^2)^2 < (R^1 + R^2)^2$$

então os retângulos se intesectam.

A função contínua e diferenciável que mede a superposição entre os retângulos e seu gradiente para esse método são:

$$f(r) = \sum_{i=1}^{nRet}\sum_{j=1}^{nCir}\sum_{k=1,k\neq i}^{nRet}\sum_{m=1}^{nCir}\max(0,(R^{ij}+R^{km})^2-(c_x^{ij}-c_x^{km})^2-(c_y^{ij}-c_y^{km})^2)^2$$

$$\nabla f(r) = \begin{pmatrix} \sum_{i=1}^{nRet}\sum_{j=1}^{nCir}\sum_{k=1,k\neq i}^{nRet}\sum_{m=1}^{nCir} -2(c_x^{ij}-c_x^{km}) \\ \sum_{i=1}^{nRet}\sum_{j=1}^{nCir}\sum_{k=1,k\neq i}^{nRet}\sum_{m=1}^{nCir} -2(c_y^{ij}-c_y^{km}) \\ \vdots \\ \vdots \end{pmatrix}$$

onde $r = (r_x^1, r_y^1, r_x^2, r_y^2, \ldots, r_x^n, r_y^n)$ e r_x^i é a coordenada x do centro do retângulo i e r_y^i é a coordenada y do centro do retângulo i. c_x^{ij} é a coordenada x do centro da circunferência j do retângulo i e c_y^{km} é a coordenada y do centro da circunferência m do retângulo k. R^{ij} é o raio da circunferência j do retângulo i e R^{km} é o raio da circunferência m do retângulo k. E $nRet$ é o número de retângulos e $nCir$ é o número de circunferências que cobrem o retângulo.

Essa função vale *zero* se não existe superposição entre as circunferências dos retângulos e qualquer valor positivo se existe superposição entre as circunferências e portanto os retângulos se intersectam.

3.2.2 Curvas de níveis dos modelos

As Figuras 28 - 31 são as curvas de níveis dos modelos contínuos apresentados na Seção 3.2.1. Os valores das funções foram normalizados para o intervalo [0;1]. Assim, a cor azul nos gráficos é onde a função vale 0 e a cor vermelha é onde a função vale 1. Para fazer esses gráficos usamos um problema que empacota 33 retângulos de dimensões $(4, 3)$ em um retângulo de dimensão $(20, 20)$. Deixamos os retângulos fixos exceto o retângulo central. Movemos esse retângulo por todas as posições do retângulo e calculamos a função.

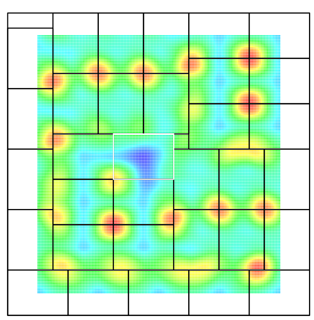


Figura 28: Curva de nível da função do modelo do vetor.

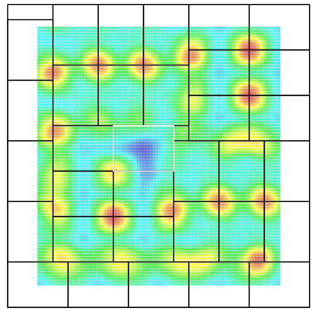


Figura 29: Curva de nível da função do modelo das restrições.

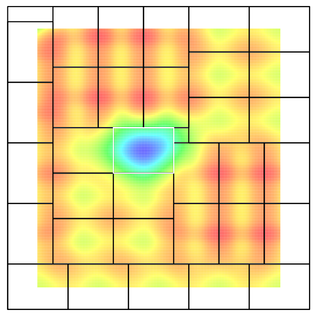


Figura 30: Curva de nível da função do modelo da distância dos centros.

3.2.3 Formulação não-linear

Queremos colocar k retângulos de dimensões (l, w) em uma região convexa de tal maneira que não ocorra superposições entre os retângulos.

Se a região convexa é um retângulo, então o objetivo é encontrar $r = (r_x^1, r_y^1, r_x^2, r_y^2, \ldots, r_x^n, r_y^n)$, onde r_x^i é a coordenada x do centro do retângulo i e r_y^i é a coordenada y do centro do retângulo i, resolvendo o problema:

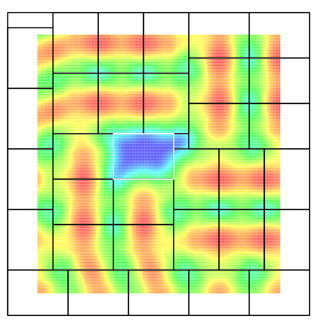


Figura 31: Curva de nível da função do modelo das circunferências.

$$
\begin{array}{ll}
\text{Minimizar} & f(r) \\
\text{sujeito a} & \\
& l(r_x^i) \leq r_x^i \leq u(r_x^i), \text{ e} \\
& l(r_y^i) \leq r_y^i \leq u(r_y^i), \text{ para } i = 1, \ldots, k.
\end{array}
$$

onde $f(r)$ é uma das funções vistas na Seção 3.2.1 e $l(x)$ é o limitante inferior de x e $u(x)$ é o limitante superior de x.

Se a região convexa, definida pela função $h(r)$, não é um retângulo, então o objetivo é encontrar $r = (r_x^1, r_y^1, r_x^2, r_y^2, \ldots, r_x^n, r_y^n)$, onde r_x^i é a coordenada x do centro do retângulo i e r_y^i é a coordenada y do centro do retângulo i, resolvendo o problema:

$$
\begin{array}{ll}
\text{Minimizar} & f(r) \\
\text{sujeito a} & \\
& h(r) \leq 0.
\end{array}
$$

Que pode ser alterada para que, caso $h(r)$ seja positiva, a função sofra uma penalidade. Assim, podemos resolver o problema:

$$\text{Minimizar} \quad f(r) + \max(0, h(r))^2$$

Se o valor da função objetiva no minimizador global desses problemas é *zero* então a resposta do problema de decisão é SIM, caso contrário, a resposta é NÃO.

Observe que os problemas descritos acima são problemas de otimização contínua onde a função objetiva tem a primeira derivada contínua.

3.2.4 Otimizador não-linear

O método descrito brevemente nessa seção lida com o minimizador de uma função suave $\varphi : I\!R^n \rightarrow I\!R$ com limites sobre as variáveis. O conjunto viável é definido por $r \in I\!R^n | l \leq r \leq u$ e o problema é:

$$\text{Minimizar } \varphi(r) \text{ sujeito a } l \leq r \leq u.$$

GENCAN [5] é um método de conjuntos ativos introduzido recentemente para suavizar a minimização de restrições de caixa. Para uma descrição de técnicas básicas de otimização contínua e métodos de conjuntos ativos veja, por exemplo, [13] e [27](pp. 326-330). GENCAN adota o critério para deixar a face de [4], que emprega os gradientes projetados espectral definidos em [8, 9]. Para a minimização "internal-to-the-face", usa um algoritmo genérico com uma busca linear que combina backtraking e extrapolação. GENCAN usa, para a direção escolhida em cada passo dentro de uma face, uma aproximação de truncamento de Newton. Isso significa que o vetor de busca é uma minimizador aproximado da aproximação quadrática da função na face corrente. Gradientes conjugados são usados para encontrar essa direção. O método é descrito completamente em [5] onde extensivos experimetos numéricos avaliam sua confiabilidade.

3.2.5 Resolvendo o problema de decisão

O método descrito na subseção anterior encontra pontos estacionários de primeira ordem (minimizadores locais) do problema. Para aumentar a probabilidade de encontrar um minimizador global, o método começa de pontos iniciais aleatórios. Com probabilidade 1, eventualmente, o método começará de um chute inicial na bacia de convergência de um minimizador global. Apesar de tudo, esse arcabouço tem dois incovenientes: (i) não queremos esperar por um tempo inifinito; e (ii) a menos que conhecemos a priori o custo ótimo do problema no minimizador global, não somos capazes de distinguir o minimizador global de um ponto estacionário.

Na prática, rodamos o método começando de N pontos iniciais diferentes. Se a solução com o custo ótimo igual a zero é encontrado então a resposta para o problema de decisão é SIM, caso contrário a resposta é "não sabemos" e assumimos que a resposta é NÃO. O fluxograma na Figura 32 mostra essa estratégia.

3.3 Empacotando o número máximo de retângulos possíveis

Na seção anterior descrevemos uma estratégia para responder a questão se é possível empacotar k retângulos em uma região convexa ou não. Para empacotar o número máximo de retângulos possíveis, vamos fazer o seguinte.

Começamos tentando empacotar apenas um retângulo e perguntamos para o problema de decisão se é possível ou não. Enquanto a resposta é SIM, tentamos com mais um retângulo. Se a resposta é NÃO, paramos. O fluxograma da Figura 33 mostra esse esquema. Na figura, n é o número de retângulos que foi possível empacotar. Note que não é necessário começar tentando empacotar um retângulo se é conhecido que a resposta para o problema de decisão, digamos, $k = k$ é SIM. Nesses casos, basta começar de $k + 1$.

3.4 Experimentos Numéricos

O programa foi codificado primeiramente em C e depois foi traduzido para Fortran77. Usamos a opção do compilador "-O4".

Olhando para o minimizador global do problema de decisão (veja Figura 32) usamos $N = 100000$, que significa que rodamos o método de 100000 pontos iniciais aleatórios ou até encontrarmos um minimizador global (detectado pelo seu custo ótimo igual a zero).

Para cada tentativa t (veja Figura 32) os pontos iniciais foram gerados como se segue. Pontos aleatórios $r^i \in I\!R^2$ foram gerados dentro da região convexa do problema. Os números aleatórios foram gerados na seguinte ordem: $r_x^1, r_y^1, r_x^2, r_y^2, \ldots$. O gerador de números aleatórios de Schrage [39] (versão double precision) com semente t foi usado para gerar números aleatórios independemente de máquina.

Resolvemos alguns problemas usando diferentes métodos:

- usando o método do vetor;
- usando o método das restrições;
- usando o método da distância entre os centros;

- usando o método do vetor com uma configuração inicial gerada pelo método das circunferências;
- usando o método das restrições com uma configuração inicial gerada pelo método das circunferências;
- usando o método da distância entre os centros com uma configuração inicial gerada pelo método das circunferências;

Para o método das circunferências, cobrimos os retângulos de diveras maneiras:

- usando o que foi descrito, tentando cobrir totalmente o retângulo;
- colocamos 6 circunferências: 4 circunferências de raio 0, um em cada vértice e 2 circunferências de raio $w/2$, um em cada lado do retângulo, onde w é o menor lado do retângulo;
- colocamos 2 circunferências de raio $w/2$, um em cada lado do retângulo, onde w é o menor lado do retângulo;
- colocamos apenas 1 circunferência de raio $w/2$, no centro do retângulo, onde w é o menor lado do retângulo

Com os resultados gerados, concluímos que desses métodos, o que resultou em melhores resultados foi o método da distância entre os centros dos retângulo com uma configuração inicial gerada pelo método das circunferências usando apenas uma circunferência.

A partir dessa conclusão, resolvemos todos os problemas de empacotar retângulos em regiões convexas que foram resolvidas em [6] e mais alguns exemplos usando esse método. Os problemas estão definidos nas Tabelas 4 - 6. Nas tabelas são dados o número do problema, a região convexa, a dimensão do retângulo, o número de retângulos que foi possível empacotar em [6] e usando modelos contínuos. As soluções são dadas nas Figuras 1-34.

3.5 Conclusões

Essa segunda parte do trabalho apresentamos uma metodologia, baseado em um problema de decisão não-linear, para resolver o problema de empacotamento de retângulos em uma região convexa. Os resultados numéricos mostram que essa metodologia é uma aproximação promissora.

Os resultados mostram que para alguns problemas resultou em melhores resultados (empacotou mais retângulos) em relação as soluções encontradas em [6], que empacota retângulos inclinados.

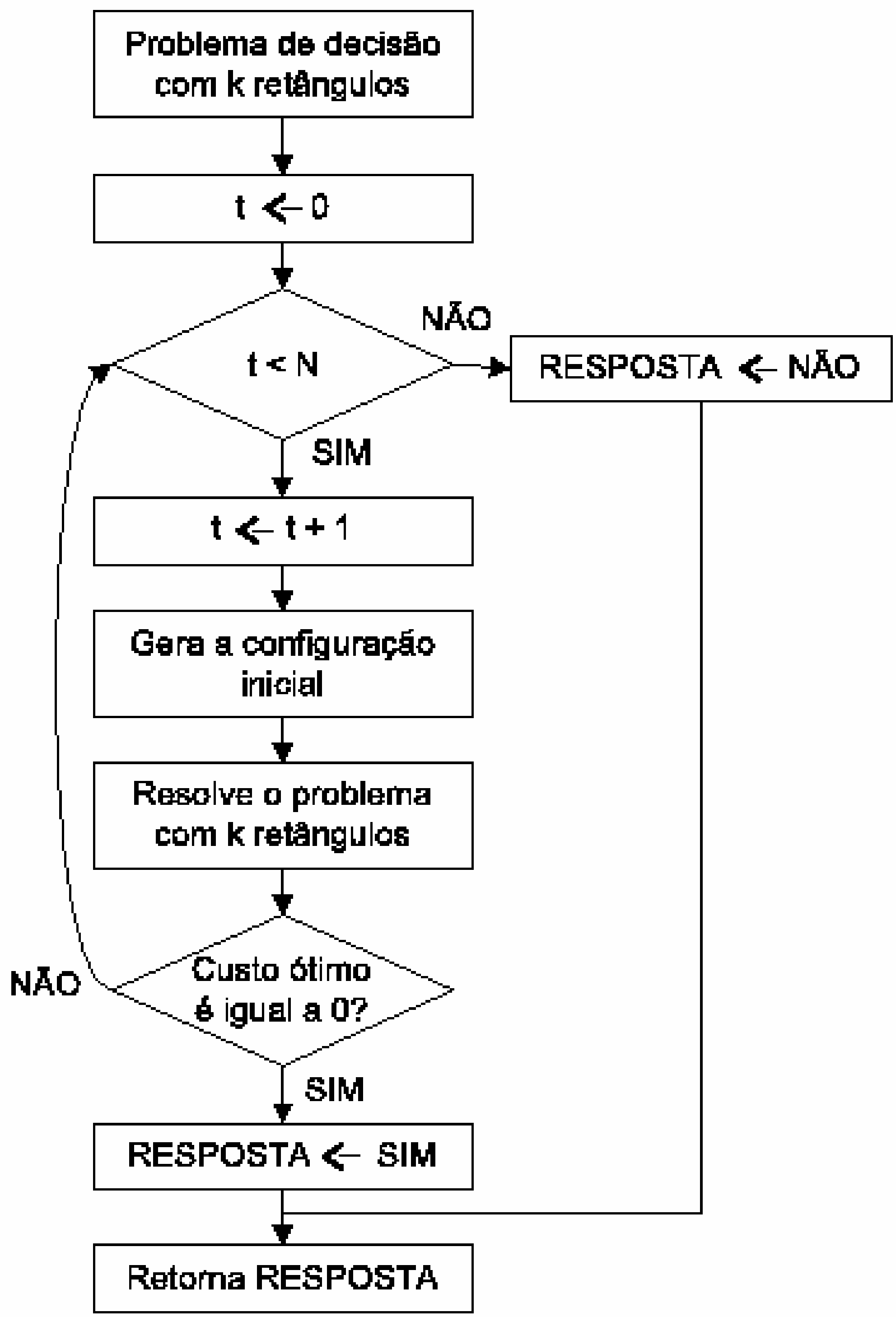


Figura 32: Resolvendo o problema de decisão.

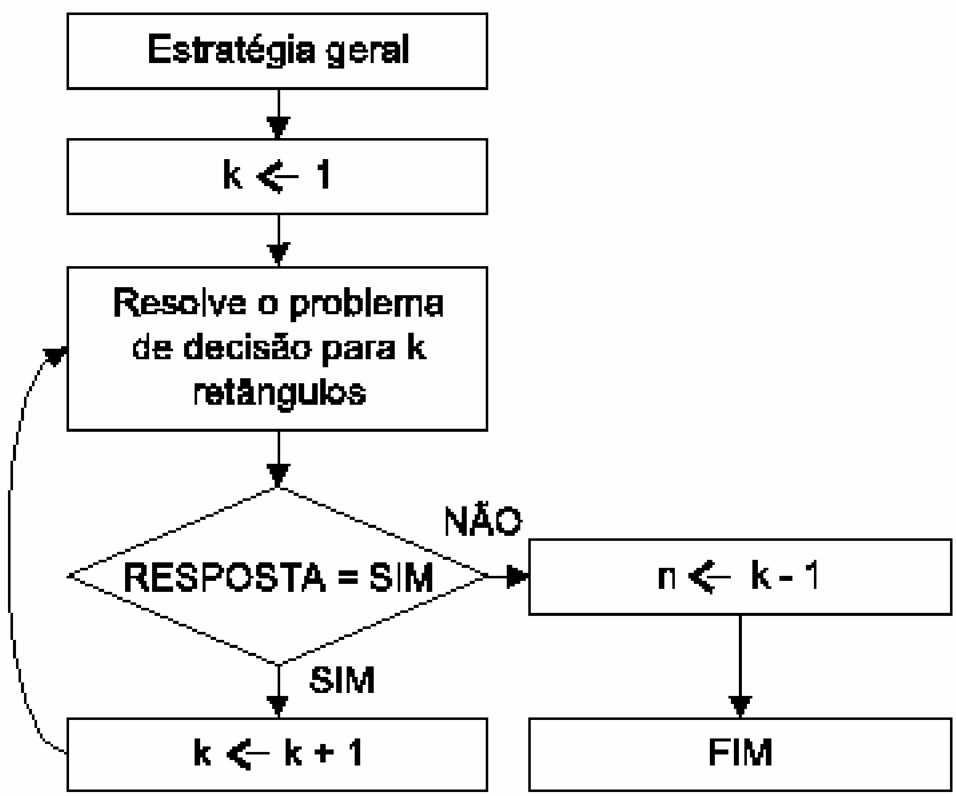


Figura 33: Esquema geral para o empacotamento de retângulos.

Problema	Região	Objetos empacotados		
		Dimensões	nRet em [6]	nRet
1	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 10$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 10$ $g_5(x_1,x_2) = x_1^2 + x_2^2 - 100$ $g_6(x_1,x_2) = -x_1 - x_2 + 3$	2×1	30	31
2	$g_1(x_1,x_2) = x_1^2 - x_2$ $g_2(x_1,x_2) = x_1^2/4 + x_2 - 5$	1×0.4	26	26
3	$g_1(x_1,x_2) = -7x_1 + 6x_2 - 24$ $g_2(x_1,x_2) = 7x_1 + 6x_2 - 108$ $g_3(x_1,x_2) = -x_2 + 4$ $g_4(x_1,x_2) = (x_1-6)^2/9 + (x_2-8)^2/9 - 1$	1.1×0.55	30	28
4	$g_1(x_1,x_2) = x_1^2 - x_2$ $g_2(x_1,x_2) = -x_1 + x_2^2 - 6x_2 + 6$ $g_3(x_1,x_2) = x_1 + x_2 - 6$	0.9×0.3	30	32
5	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 8$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 10$ $g_5(x_1,x_2) = (x_1-6)^2 + (x_2-0)^2 - 81$ $g_6(x_1,x_2) = (x_1-1.7)^2 + (x_2-10)^2 - 81$	2×0.6	34	38
6	$g_1(x_1,x_2) = -x_1 + x_2^2 - 6x_2 + 6$ $g_2(x_1,x_2) = x_1 + x_2^2 - 3x_2 - 3/4$	0.9×0.3	31	30
7	$g_1(x_1,x_2) = (x_1-2)^2/4 + (x_2-4)^2/16 - 1$	2×0.5	19	19
8	$g_1(x_1,x_2) = (x_1-3)^2/4 + (x_2-4)^2/16 - 1$ $g_2(x_1,x_2) = (x_1-2.65)^2/2^2 + (x_2-4)^2/4^2 - 1$ $g_3(x_1,x_2) = -x_1 + 1$ $g_4(x_1,x_2) = x_1 - x_2 - 1$ $g_5(x_1,x_2) = x_1 + x_2 - 9$	0.8×0.6	24	22
9	$g_1(x_1,x_2) = (x_1-6)^2/4 + (x_2-6)^2/36 - 1$ $g_2(x_1,x_2) = (x_1-6)^2/36 + (x_2-6)^2/4 - 1$ $g_3(x_1,x_2) = x_1 - x_2 - 3$ $g_4(x_1,x_2) = -x_1 + x_2 - 2$	0.7×0.5	30	32
10	$g_1(x_1,x_2) = (x_1-6)^2/36 + (x_2-6)^2/4 - 1$ $g_2(x_1,x_2) = (x_1-6)^2/9 + (x_2-8)^2/9 - 1$	0.95×0.35	30	34
11	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - (2+\sqrt{2}/2)$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - (2+\sqrt{2}/2)$	1×1	5	4
12	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - (3+\sqrt{2}/2)$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - (3+\sqrt{2}/2)$	1×1	10	9
13	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 3.878$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 3.878$	1×1	11	9
14	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 4.676$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 4.676$	1×1	17	16
15	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - (7+\sqrt{7}/2)$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - (7+\sqrt{7}/2)$	1×1	18	16

Tabela 4: Problemas e regiões.

Problema	Região	Obejtos empacotados		
		Dimensões	nRet em [6]	nRet
16	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - (3+4\sqrt{2}/3)$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - (3+4\sqrt{2}/3)$	1×1	19	16
17	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - (7/2+3\sqrt{2}/2)$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - (7/2+3\sqrt{2}/2)$	1×1	26	25
18	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 6.621$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 6.621$	1×1	37	36
19	$g_1(x_1,x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(3/2+\sqrt{3})$ $g_2(x_1,x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1,x_2) = -x_2$	1×1	27	25
20	$g_1(x_1,x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(2+4/\sqrt{3})$ $g_2(x_1,x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1,x_2) = -x_2$	1×1	28	26
21	$g_1(x_1,x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(3+4/\sqrt{3})$ $g_2(x_1,x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1,x_2) = -x_2$	1×1	29	29
22	$g_1(x_1,x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(2+2\sqrt{3})$ $g_2(x_1,x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1,x_2) = -x_2$	1×1	30	29
23	$g_1(x_1,x_2) = \sqrt{3}x_1 + x_2 - \sqrt{3}(4+4/\sqrt{3})$ $g_2(x_1,x_2) = -\sqrt{3}x_1 + x_2$ $g_3(x_1,x_2) = -x_2$	1×1	31	30
24	$g_1(x_1,x_2) = (x_1/6)^4 + (x_2/2)^4 - 1$ $g_2(x_1,x_2) = 8x_1 - 11x_2 - 26$	1.9×0.5	27	30
26	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 10$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 8$	2×1	40	40
27	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 23$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 21$	6×5	16	16
28	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 10$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 8$	2×1	34 + 6 fixos	34 + 6 fixos
29	$g_1(x_1,x_2) = -x_1$ $g_2(x_1,x_2) = x_1 - 10$ $g_3(x_1,x_2) = -x_2$ $g_4(x_1,x_2) = x_2 - 8$	2×1	34 + 6 fixos	34 + 6 fixos

Tabela 5: Problemas e regiões (continuação).

Problema	Região	Obejtos empacotados		
		Dimensões	nRet em [6]	nRet
31	$g_1(x_1, x_2) = -x_1 + x_2 - 6$ $g_2(x_1, x_2) = x_1 - x_2 - 6$ $g_3(x_1, x_2) = -x_1 - x_2 + 6$ $g_4(x_1, x_2) = x_1 + x_2 - 18$ $g_5(x_1, x_2) = -x_1 + 1$ $g_6(x_1, x_2) = x_1 - 11$ $g_7(x_1, x_2) = -x_2 + 1$ $g_8(x_1, x_2) = x_2 - 11$	2×1	28 + 2 fixos	28 + 2 fixos
33	$g_1(x_1, x_2) = -x_1 + x_2 - 6$ $g_2(x_1, x_2) = x_1 - x_2 - 6$ $g_3(x_1, x_2) = -x_1 - x_2 + 6$ $g_4(x_1, x_2) = x_1 + x_2 - 18$ $g_5(x_1, x_2) = -x_1 + 1$ $g_6(x_1, x_2) = x_1 - 11$ $g_7(x_1, x_2) = -x_2 + 1$ $g_8(x_1, x_2) = x_2 - 11$	2×1	24 + 6 fixos	24 + 6 fixos
34	$g_1(x_1, x_2) = -x_1 + x_2 - 6$ $g_2(x_1, x_2) = x_1 - x_2 - 6$ $g_3(x_1, x_2) = -x_1 - x_2 + 6$ $g_4(x_1, x_2) = x_1 + x_2 - 18$ $g_5(x_1, x_2) = -x_1 + 1$ $g_6(x_1, x_2) = x_1 - 11$ $g_7(x_1, x_2) = -x_2 + 1$ $g_8(x_1, x_2) = x_2 - 11$	2×1	20 + 10 fixos	20 + 10 fixos

Tabela 6: Problemas e regiões (continuação).

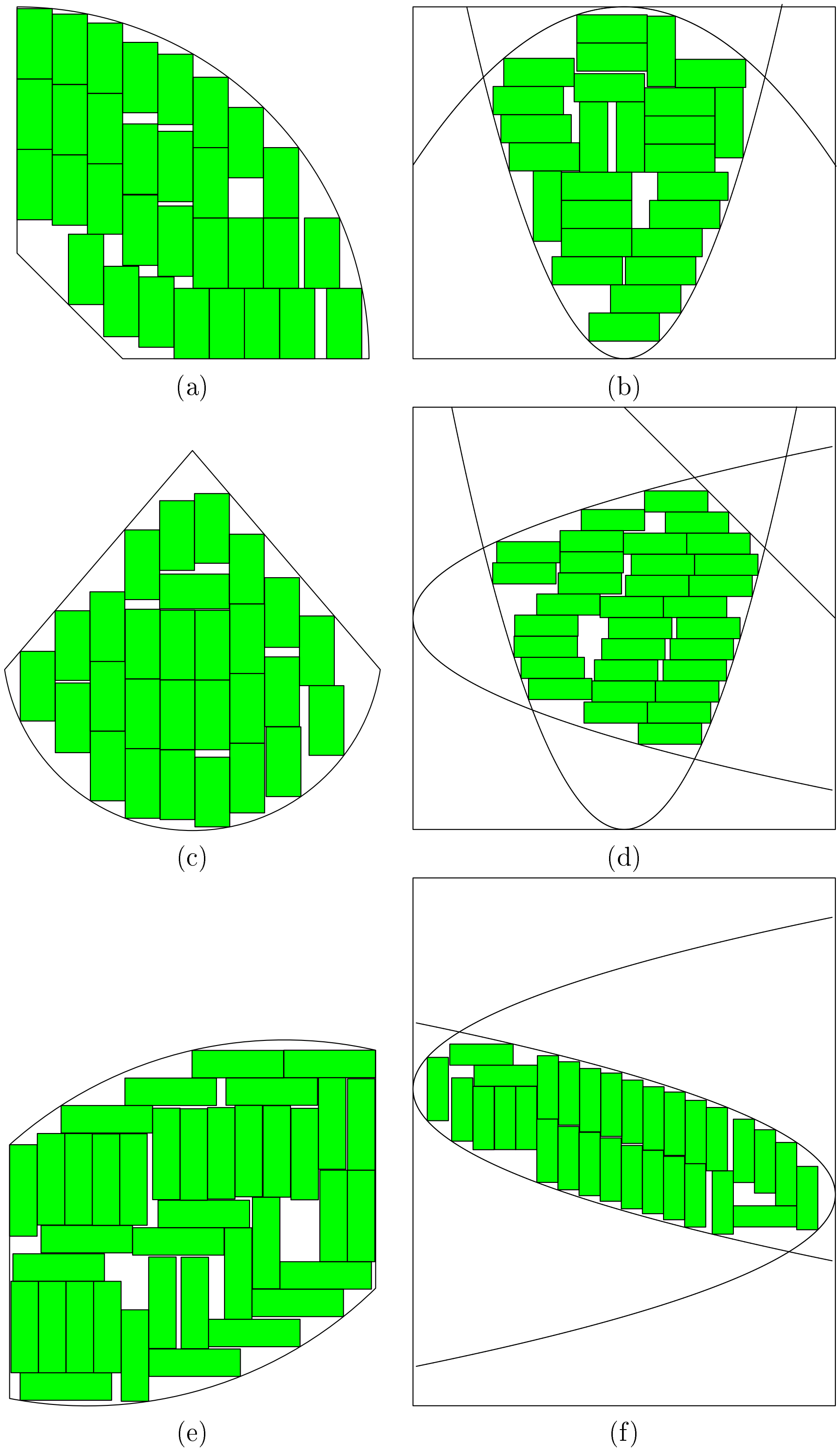


Figura 34: (a–c) Restrições lineares e círculos, (d–f) Restrições lineares e quadráticos (Problemas 1 a 6).

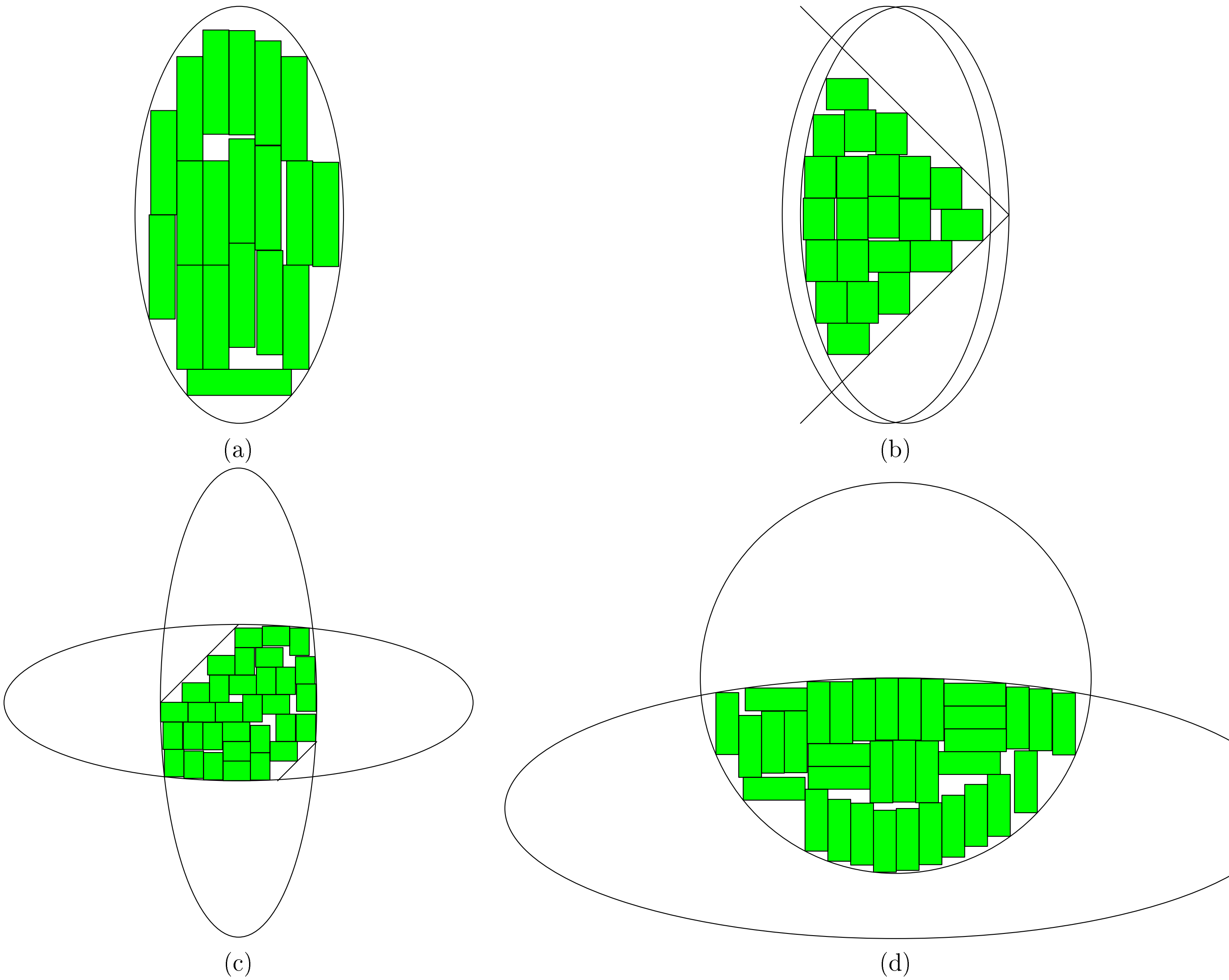
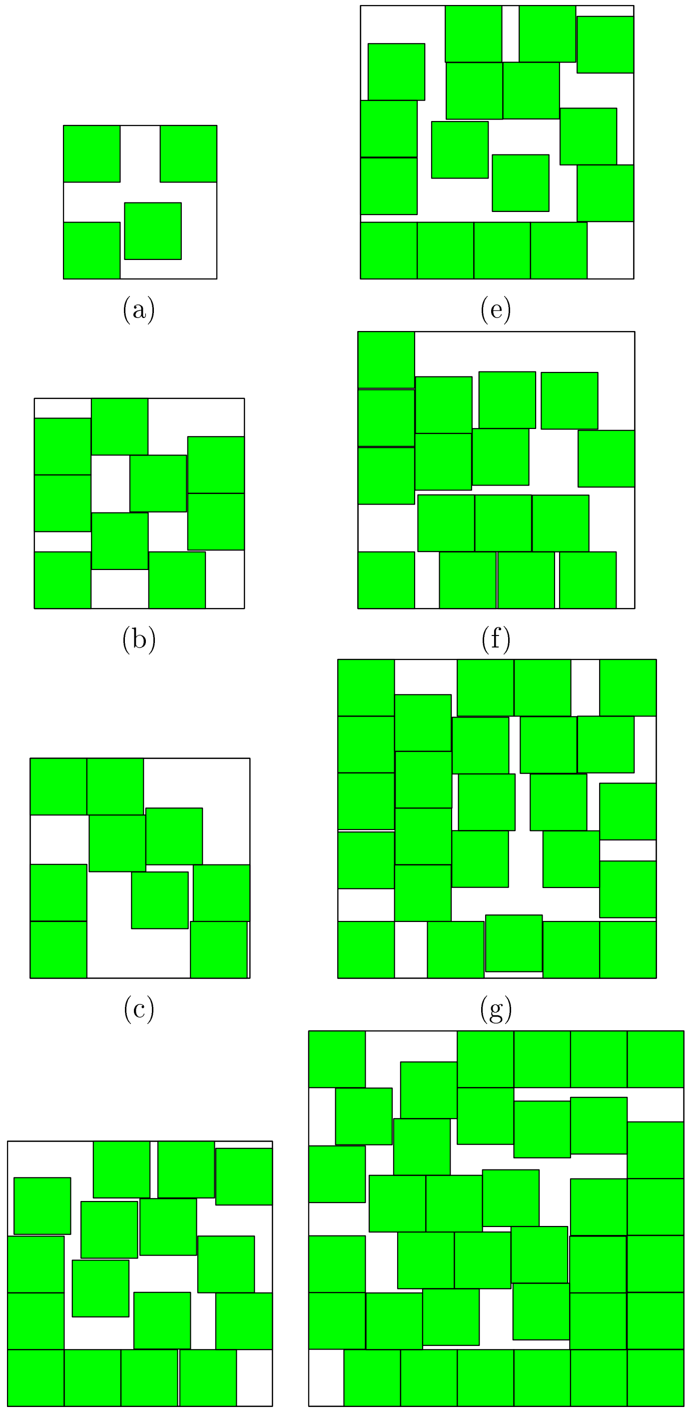


Figura 35: Restrições lineares e elipses (Problemas 7 a 10).



(a) (e) (b) (f) (c) (g) (d) (h)

Figura 36: Quadrados em quadrados (Problemas 11 a 18).

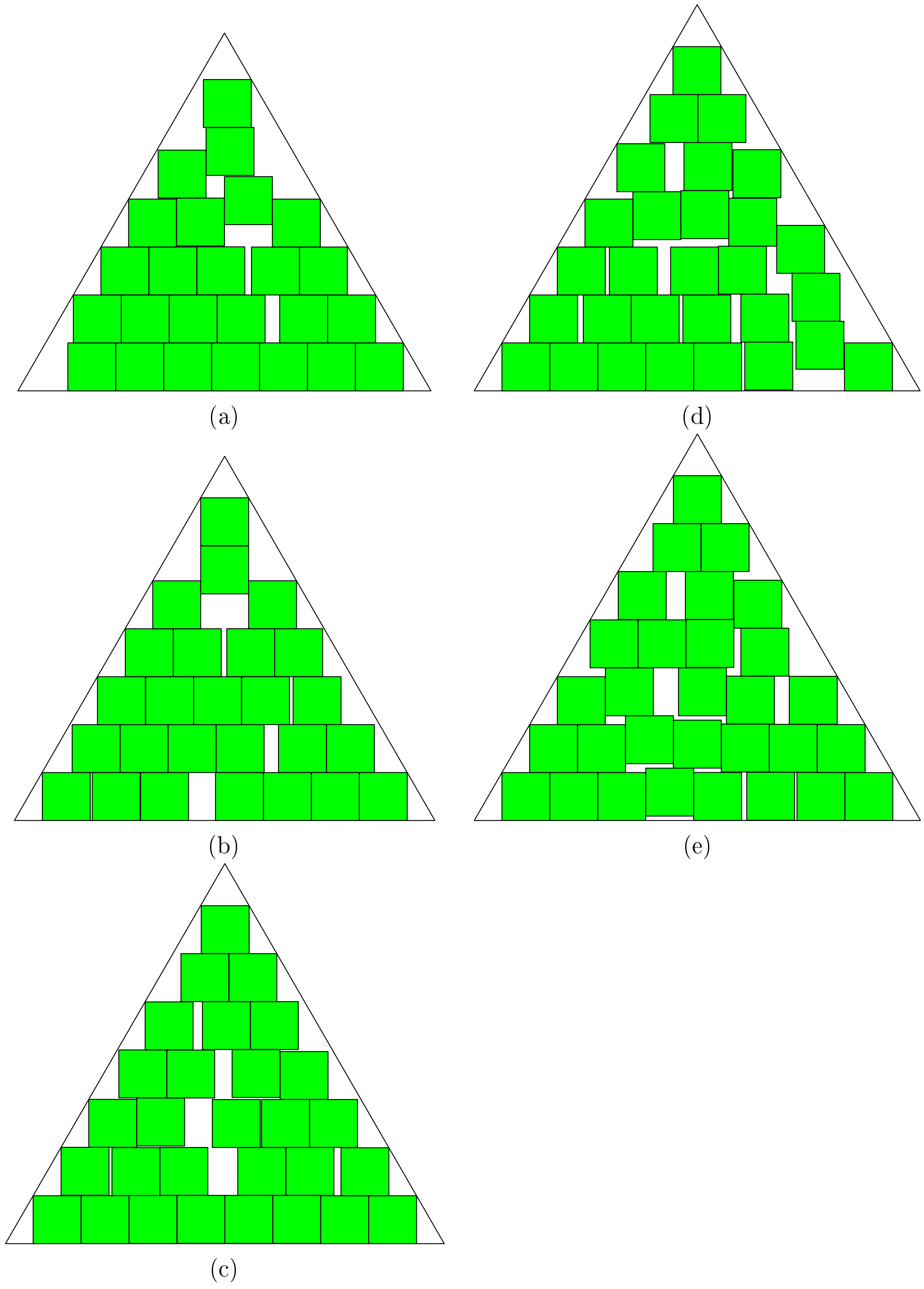


Figura 37: Quadrados em triângulos (Problemas 19 a 23).

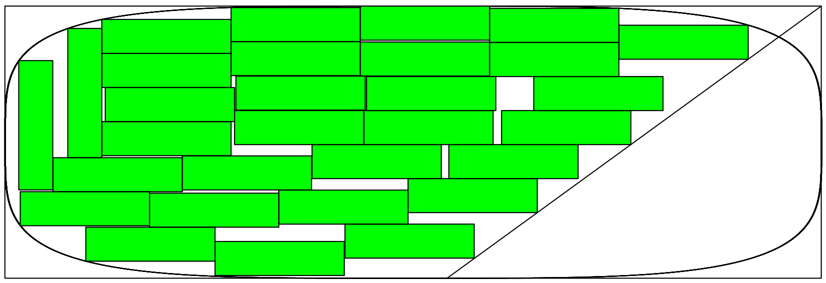
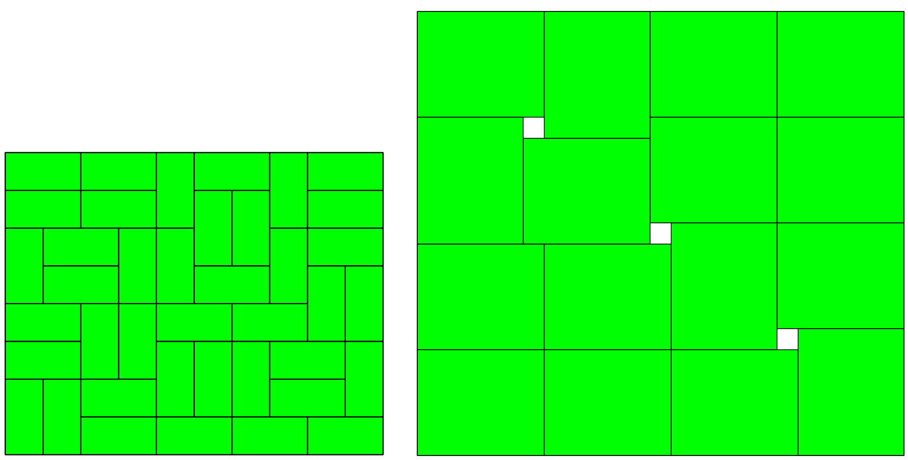
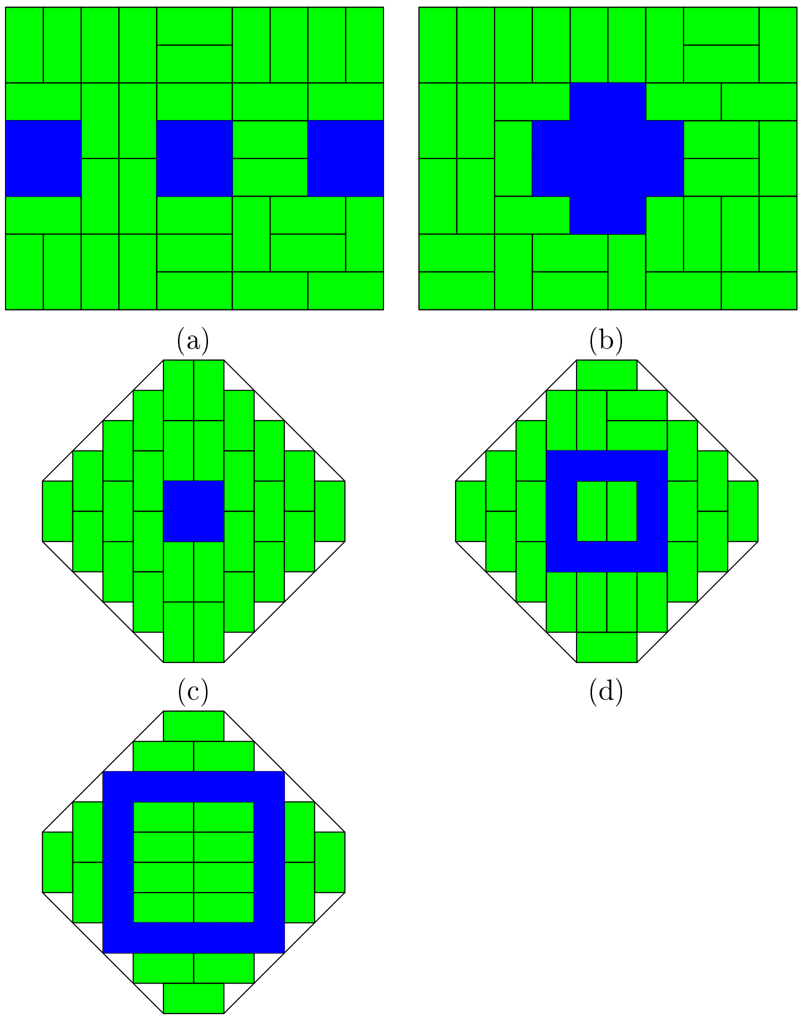


Figura 38: Curvas e restrições lineares (Problema 24).



(a) (b)

Figura 39: Retângulo em retângulo (Problemas 26 e 27).



(a) (b) (c) (d) (e)

Figura 40: Regiões proibidas para configurações especias em paletes (Problemas 28 a 34).

4 Conclusões

Na primeira parte do trabalho estudamos o algoritmo L para o empacotamento de retângulos introduzido em [26]. Em [26] o tempo de execução é apontado como a maior deficiência do método. Em função disso, na nossa implementação tentamos, utilizando mais memória, diminuir o tempo de execução do algoritmo.

Na segunda parte do trabalho apresentamos uma metodologia, baseado em um problema de decisão não-linear, para resolver o problema de empacotamento de retângulos em uma região convexa. Os resultados numéricos mostram que essa metodologia é uma aproximação promissora.

Nas duas abordagens, os retângulos tem dimensões iguais e os retângulos são empacotados ortogonalmente em relação aos eixos.

O algoritmo L empacota retângulos apenas em retângulos, enquanto usando modelos contínuos conseguimos empacotar retângulos em qualquer região convexa. Mas, em compensação, o algoritmo L encontra soluções mais rápidas do que usando os modelos contínuos, para o empacotamento de retângulos em retângulos.

No algoritmo L conseguimos diminuir o tempo de execução, em relação a [26], para encontrar uma solução e usando os modelos contínuos conseguimos encontrar algumas solução onde pode ser colocado mais retângulos do que em [6].

Lista de Figuras

Lista de Tabelas

Referências

[1] M. Arenales, R. Morabito e H. Yanasse (eds.) (1999). Special issue: Cutting and packing problems. *Pesquisa Operacional* 19, pp. 109–284.

[2] R. Balasubramanian (1992). The pallet loading problem: A survey. *International Journal of Production Economics* 28, pp. 217–225.

[3] S. Bhattacharya, R. Roy e S. Bhattacharya (1998). An exact depth-first algorithm for the pallet loading problem. *European Journal of Operational Research* 110, pp. 610–625.

[4] E. G. Birgin, J. M. Martínez (2001). A box constrained optimization algorithm with negative curvature directions and spectral projected gradients. Computing [Suppl] 15, pp. 49–60.

[5] E. G. Birgin, J. M. Martínez (2002). Large-scale active-set box-constrained optimization method with spectral projected gradients. Computational Optimization and Applications 23, pp. 101–125.

[6] E.G. Birgin, J.M. Martínez, W.F.Mascarenhas e D.P. Ronconi (2003). Method of Sentinels for Packing Objects on Arbitrary Regions.

[7] E.G. Birgin, J.M. Martínez e D.P. Ronconi (2003). Optimizing the Packing of Cylinders into a Rectangular Container: A Nonlinear Approach. Submetido a *European Journal of Operational Research.*

[8] E. G. Birgin, J. M. Martínez, M. Raydan (2000). Nonmonotone spectral projected gradient methods on convex sets. SIAM Journal on Optimization 10, pp. 1196–1211.

[9] E. G. Birgin, J. M. Martínez, M. Raydan (2001). SPG: Software for convex-constrained optimization. ACM Transactions on Mathematical Software 27, pp. 340–349.

[10] E. Bischoff e W. Dowsland (1982). An application of the micro to product design and distribution. *Journal of the Operational Research* 33, pp. 271–280.

[11] E. Bischoff e G. Waescher (eds.) (1995). Special issue on cutting and packing. *European Journal of Operational Research* 84, pp. 503–712.

[12] T. Cormen, C. Leiserson e R. Rivest (1990). *Introduction to algorithms.* The MIT Press, McGraw-Hill Book Company.

[13] J. E. Dennis Jr., R. B. Schnabel (1983). Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Englewoods Cliffs.

[14] K. Dowsland (1987). An exact algorithm for the pallet loading problem. *European Journal of Operational Research* 31, pp. 78–84.

[15] K. Dowsland e W. Dowsland (1992). Packing problems. *European Journal of Operational Research* 56, pp. 2–14.

[16] H. Dyckhoff (1990). A typology of cutting and packing problems. *European Journal of Operational Research* 44, pp. 145–159.

[17] H. Dyckhoff e U. Finke (1992). *Cutting and packing in production and distribution: Typology and bibliography.* Springler-Verlag Co: Heildelberg.

[18] H. Dyckhoff, G. Scheithauer e J. Terno (1997). Cutting and packing. In M. Amico, F. Maffioli, F., S. Martello (ed). *Annotated bibliographies in combinatorial optimisation*. John Wiley & Sons: New York, pp. 393–414.

[19] Y. G e M. Kang (2001). A fast algorithm for two-dimensional pallet loading problems of large size. *European Journal of Operational Research* 134, pp. 193–200.

[20] A. Herbert e K. Dowsland (1996). A family of genetic algorithms for the pallet loading problem. *Annals of Operations Research* 63, pp. 415–436.

[21] M. Hifi (ed.) (2002). Special issue on cutting and packing. *Studia Informatica Universalis* 2, pp. 1–161.

[22] T. Hodgson (1982). A combined approach to the pallet loading problem. *IIE Transactions* 14, pp. 176–182.

[23] A. Letchford e A. Amaral (2001). Analysis of upper bounds for the pallet loading problem. *European Journal of Operational Research* 132, pp. 582–593.

[24] L. Lins, S. Lins e S. Melo (2002). PHORMA: Symmetry robust memory management of multidimensional arrays. Submetido a *Discrete Applied Mathematics*.

[25] L. Lins, S. Lins e R. Morabito (2002). An n-tet graph approach for nonguillotine packings of n-dimensional boxes into an n-container. *European Journal of Operational Research* 141, pp. 421–439.

[26] L. Lins, S. Lins e R. Morabito (2003), An L-Approach for Packing (l,w)-Rectangles into Rectangular and L-Shaped Pieces. *Journal of the Operational Research Society.*

[27] D. G. Luenberger (1984). Linear and Nonlinear Programming. Addison-Wesley, California, London, Amsterdam, Ontario, Sydney.

[28] S. Martello (ed.) (1994), Special issue: Knapsack, packing and cutting, Part II: Multidimensional knapsack and cutting stock problems. *INFOR*, 32.

[29] R. Morabito e M. Arenales (1995). An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems. *European Journal of Operational Research* 84, pp. 599–617.

[30] R. Morabito e R. Farago (2002). A tight Lagrangean relaxation bound for the manufacturer's pallet loading problem. *Studia Informatica Universalis* 2, pp. 57–76.

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

[32] R. Morabito e S. Morales (1999). Errata: A simple and effective recursive procedure for the manufacturers pallet loading problem . *Journal of the Operational Research Society* 50, pp. 876.

[33] J. Nelissen (1994). Solving the pallet loading problem more effciently. Working Paper, Graduiertenkolleg Informatik und Technik, Aachen.

[34] J. Nelissen (1995). How to use structural constraints to compute an upper bound for the pallet loading problem. *European Journal of Operational Research* 84, pp. 662–680.

[35] G.L. Nemhauser e L.A. Wosley (1988). Integer and Combinatorial Optimization. Wiley.

[36] G. Scheithauer e G. Sommerweiss (1998). 4-block heuristic for the rectangle packing problem. *European Journal of Operational Research* 108, pp. 509–526.

[37] G. Scheithauer e J. Terno (1996). The G4-heuristic for the pallet loading problem. *Journal of the Operational Research Society* 47, pp. 511–522.

[38] A. Smith e P. De Cani (1980). An algorithm to optimize the layout of boxes in pallets. *Journal of the Operational Research Society* 31, pp. 573–578.

[39] L. Schrage (1979). A more portable Fortran random number generator. ACM Transactions on Mathematical Software 5, pp. 132–138.

[40] H. Steudel (1979). Generating pallet loading patterns: A special case of the two-dimensional cutting stock problem. *Management Science* 10, pp. 997–1004.

[41] P. Sweeney e E. Paternoster (1992). Cutting and packing problems: A categorized, application-oriented research bibliography. *Journal of the Operational Research Society* 43, pp. 691–706.

[42] R. Tsai, E. Malstrom e W. Kuo (1993). Three dimensional palletization of mixed box sizes. *IEE Transactions* 25, pp. 64–75.

[43] P. Wang e G. Waescher (eds.) (2002). Special issue on cutting and packing. *European Journal of Operational Research* 141, pp. 239–469.