First Fit Decreasing with Backtracking: um algoritmo exato para o problema de corte unidimensional baseado na conjectura Mirup

Glauber Cintra [1]

Instituto de Matemática e Estatística — Universidade de São Paulo
Rua do Matão, 1010 — CEP 05508-900 — São Paulo, SP
E-mail: glauber@ime.usp.br

Resumo. Diversos autores têm modelado o problema de corte de estoque unidimensional (PCE_1) como um problema de programação linear inteira (PLI) e utilizado o método de geração de colunas para encontrar soluções (fracionárias) para a relaxação linear do PLI. Conjectura-se que a diferença entre o valor de uma solução inteira ótima do PCE_1 e o valor de uma solução ótima da relaxação linear é menor que 2. Temos assim um excelente limitante, assumindo que esta conjectura é válida.

Desenvolvemos então um algoritmo exato para o PCE_1, que chamamos de First Fit Decreasing with Backtracking (FFDWB), que utiliza o limitante citado acima para encontrar uma solução inteira ótima para o PCE_1. Utilizamos o FFDWB como subrotina de um outro algoritmo, baseado no método de geração de colunas, e os resultados obtidos ao resolver 4000 instâncias de teste encontradas na literatura foram os melhores do que aqueles obtidos por todos os algoritmos testados para estas mesmas instâncias. Resolvemos também centenas de instâncias reais, relacionadas ao processo de corte do aço na indústria da construção civil, tendo encontrado soluções ótimas para todas estas instâncias.

Palavras-chave: problema de corte de estoque unidimensional, geração de colunas, conjectura MIRUP.

Abstract. Several authors have modelled the unidimensional cutting stock problem (CSP_1) as a integer linear programming problem (IP) and used the column generation method to find (fractional) solutions for the IP linear relaxation. It has been conjectured that the gap between the value of an optimal integer solution of the CSP_1 and the value of an optimal solution of the IP relaxation is smaller than 2. This is an excellent bound, assuming that this conjecture holds.

We designed an exact algorithm for CSP_1, which we called First Fit Decreasing with Backtracking (FFDWB), that uses the bound cited above to find an optimal integer solution for CSP_1. We used FFDWB as a subroutine in another algorithm, based on the column generation method, and the results we obtained by solving 4000 instances from the literature were better than those obtained by all other algorithms tested with the same instances. We also solved hundreds of real instances, coming from de process of cutting of steel in the civil industry, and we found optimal solutions for all these instances.

keywords: unidimensional cutting stock problem, column generation, MIRUP conjecture.

Introdução

Nas últimas quatro décadas, os problemas de corte e empacotamento têm sido largamente estudados por um número crescente de pesquisadores, gerando e tendo se beneficiado de avanços significativos em diversas áreas, tais como *programação linear*, *programação dinâmica*, *teoria da complexidade computacional* e *algoritmos de aproximação*. O interesse por estes problemas é em parte explicado por sua aparente simplicidade e grande aplicabilidade prática. São porém, problemas de natureza complexa, *$\mathcal{NP}$-difíceis* [9].

Quanto à aplicabilidade desses problemas, especialmente nas indústrias, podemos citar os processos de corte de barras de aço e alumínio, bobinas de papel, chapas de metal e madeira, lâminas de vidro, peças de carpete, blocos de isopor, etc. Já os processos de carregamento de contêineres de navio, carrocerias de caminhões e vagões de trem são alguns exemplos práticos de problemas de empacotamento. Pequenas melhorias nestes processos podem levar a ganhos substanciais, dependendo da escala de produção, e representar uma vantagem decisiva na competição com outras empresas do setor. Neste artigo, nosso interesse concentra-se no problema de corte de estoque unidimensional (PCE_1).

O PCE_1 consiste em: dado um objeto, genericamente denominado de *barra*, de comprimento L, e uma lista de m itens, cada item i com comprimento l_i e demanda $d_i \in I\!N$ $(i = 1, \ldots, m)$, determinar o

[1] Bolsista de Doutorado do CNPq.

menor número de barras necessárias para atender a demanda. Obviamente também estamos interessados em determinar como as barras devem ser cortadas. Este problema pertence à classe 1/V/I/R segundo a classificação de Dyckhoff [6].

Este artigo está organizado da seguinte maneira. Na Seção 1 mostramos como modelar o PCE_1 e como resolver a relaxação linear do modelo usando o método de geração de colunas. Na Seção 2 introduzimos a conjectura MIRUP e sua aplicabilidade na resolução do PCE_1. Na seção 3 apresentamos um algoritmo exato para o PCE_1, baseado na conjectura MIRUP, e sua utilização como subrotina de outro algoritmo para o PCE_1. Na Seção 4 discutimos os resultados de testes computacionais e na última seção tecemos algumas considerações finais.

1 O Método de Geração de Colunas

Como é bem conhecido, o PCE_1 pode ser formulado como um problema de programação linear inteira (PLI). Descrevemos a seguir como fazer isso. Primeiramente, representamos cada possível forma de cortar uma barra, o que chamamos de *padrão de corte* (ou simplesmente *padrão*), por um vetor-coluna. Supondo que haja m itens, cada padrão j é representado por um vetor-coluna a_j, com m elementos, cujo i-ésimo elemento indica o número de vezes que o item i ocorre nesse padrão. Dizemos que um padrão é *viável* se $\sum_{i=1}^{m}(a_j)_i l_i \leq L$. O problema agora consiste em considerar os n padrões viáveis e decidir quantas vezes cada padrão deve ser cortado de modo a atender a demanda, minimizando o total de barras utilizadas.

Para isso, introduzimos uma variável vetor x no qual cada elemento x_j $(j = 1, \ldots, n)$, indica quantas vezes o padrão j deve ser cortado (obviamente $x_j \in I\!N$). Note que, se x é uma solução viável do problema, então o valor $\sum_{j=1}^{n} x_j$ corresponde ao número de barras a serem cortadas. Assim, denotando por A a matriz $m \times n$ cujas colunas são os padrões viáveis, e representando por d o vetor das demandas, o problema pode ser formulado como:

$$\begin{array}{ll} \min \sum_{j=1}^{n} x_j & \\ \text{sujeito a } Ax = d & (1) \\ x_j \geq 0 \text{ e inteiro} \qquad j = 1, \ldots, n. & \end{array}$$

A formulação (1) traz consigo duas dificuldades em termos computacionais. A primeira é determinar a matriz A (que pode ter um número exponencial de colunas); a segunda é resolver um problema de programação linear inteira (que é sabido ser $\mathcal{NP}$-difícil). Para lidar com estas dificuldades, Gilmore e Gomory propuseram o método de geração de colunas [13, 14]. A idéia do método de geração de colunas consiste, como o próprio nome sugere, em ir gerando gradativamente as colunas da matriz A (dos padrões viáveis). Iniciamos com a matriz A correspondendo a uma solução básica viável. Com essa matriz A, que chamaremos de *base*, resolvemos a relaxação linear de (1) usando o algoritmo *simplex revisado* [3], onde a cada iteração uma nova coluna é gerada resolvendo-se um problema da mochila. Este algoritmo é conhecido como *simplex revisado com geração de colunas*, que chamaremos simplesmente de SimplexGC, descrito a seguir.

Algoritmo SimplexGC

Entrada: $(L, l_1, \ldots, l_m, d_1, \ldots, d_m)$.
Saída: Uma solução ótima da relaxação linear de (1).

1 Para $i = 1$ até m faça
  1.1 Para $j = 1$ até m se $i = j$ então faça $a_{ij} = \lfloor \frac{L}{l_i} \rfloor$; caso contrário, faça $a_{ij} = 0$.
2 Faça $x_i = \frac{d_i}{a_{ii}}$ $(i = 1, \ldots, m)$.
3 Resolva $yA = c$.
4 Execute o algoritmo MOCHILA com parâmetros $L, l_1, \ldots, l_m, y_1, \ldots, y_m$.
5 Se $\sum_{i=1}^{m} y_i z_i \leq 1$, retorne x e pare.
6 Caso contrário, resolva $Ab = z$.
7 Calcule $t = min\{\frac{x_i}{b_i} \mid b_i > 0 \ (i = 1, \ldots, m)\}$ e $s = min\{i \mid \frac{x_i}{b_i} = t \ (i = 1, \ldots, m)\}$.
8 Para $i = 1$ até m faça $a_{is} = z_i$ $(i = 1, \ldots, m)$.
  8.1 Se $i = s$ então faça $x_i = t$; caso contrário, faça $x_i = x_i - b_i t$.
9 Retorne ao passo 3.

Eis a descrição do algoritmo MOCHILA, usado no algoritmo SimplexGC.

Algoritmo MOCHILA

Entrada: $(L, l_1, \ldots, l_m, y_1, \ldots, y_m)$.
Saída: $z_1, \ldots, z_m \in I\!N$ tais que $\sum_{i=1}^{m} l_i z_i \leq L$ e $\sum_{i=1}^{m} y_i z_i$ é máximo.

1 Classifique os itens em ordem decrescente de custo relativo $(\frac{y_i}{l_i})$.
2 Faça $z_j = \lfloor (L - \sum_{i=1}^{j-1} l_i z_i)/l_j \rfloor$ para $j = 1, \ldots, m$, $z^* = z$ e $M = \sum_{i=1}^{m} y_i z_i^*$.
3 Faça $k = max\{i \mid z_i > 0 \ e \ \sum_{j=1}^{i-1} y_j z_j \ + \ y_i(z_i - 1) \ + \ \frac{y_{i+1}}{l_{i+1}}(L - \sum_{j=1}^{i-1} l_j z_j - l_i(z_i - 1)) > M \ (i = m-1, \ldots, 1)\}$.
3.1 Se não existe tal k então devolva z^* e pare.
3.2 Caso contrário faça $z_k = z_k - 1$ e $z_j = \lfloor (L - \sum_{i=1}^{j-1} l_i z_i)/l_j \rfloor$, para $j = k+1, \ldots, m$.
4 Se $M \leq \sum_{i=1}^{m} y_i z_i$ faça $z^* = z$ e $M = \sum_{i=1}^{m} y_i z_i^*$.
5 Retorne ao passo 3.

O SimplexGC fornece uma solução ótima, não necessariamente inteira. Para contornar este problema, Gilmore e Gomory [13] propuseram arredondar para cima o valor das variáveis, após achar uma solução ótima, obtendo assim uma solução inteira. No entanto, este procedimento usualmente acarreta a produção de itens em quantidade superior à demanda, e conduz a uma solução inteira eventualmente longe da ótima. Recentemente, Wäescher [26] propôs arredondar para baixo o valor das variáveis, após achar uma solução ótima e então tratar o problema residual conforme descrevemos a seguir.

Aplicando o SimplexGC obtemos uma solução ótima x para a relaxação linear de (1). Chamemos de v_{rl} o valor desta solução. O método proposto para encontrar uma solução inteira de (1) consiste em arredondar x para baixo obtendo $\bar{x}$, ou seja, $\bar{x}_j = \lfloor x_j \rfloor$ $(j = 1, \ldots, m)$. Seja $\bar{v}$ o valor da solução $\bar{x}$. Claramente $\bar{v} \leq v_{rl}$. Se $\bar{v} = v_{rl}$, então x é uma solução inteira ótima, caso contrário, alguma parte da demanda não foi atendida. Temos assim um problema residual onde cada item i possui demanda inteira $d'_i = d_i - \sum_{j=1}^{m} a_{ij}\bar{x}_j$ $(i = 1, \ldots, m)$. Após calcular $\bar{x}$ e d', se $\bar{v} > 0$ então aplicamos recursivamente o algoritmo ao problema residual; caso contrário, resolvemos novamente o problema utilizando um algoritmo que forneça uma solução inteira. Na Seção 4 descrevemos como resolver uma instância com $\bar{v} = 0$, utilizando v_{rl}, fornecido pelo SimplexGC, que é um excelente indicador do valor de uma solução inteira ótima, como veremos na seção seguinte.

2 A Conjectura MIRUP

Seja I uma instância de um problema de programação linear inteira P, no qual queremos minimizar a função objetivo. Seja $v(I)$ o valor de uma solução inteira ótima de I e $v_{rl}(I)$ o valor de uma solução ótima da relaxação linear do problema. Baum e Trotter [1] definiram a *Integer Round-Up Property (IRUP)* da seguinte forma:

Definição 2.1. *Um problema de programação linear inteira P (de minimização) possui a integer round-up property (IRUP) se $v(I) = \lceil v_{rl}(I) \rceil$ para toda instância $I \in P$.*

Também dizemos que uma instância I de um PLI possui a IRUP se $v(I) = \lceil v_{rl}(I) \rceil$. Baum e Trotter [1] estabeleceram que esta propriedade é válida para certas classes de matrizes que surgem no contexto da teoria dos polimatróides. Em 1985, Marcotte [16] mostrou que várias classes de instâncias do PCE_1 possuem a IRUP, no entanto, Dyer, Frieze e McDiarmid [17] provaram que é $\mathcal{NP}$-difícil determinar se uma instância qualquer do PCE_1 possui ou não a IRUP.

Em 1986 Marcotte [17] apresentou uma instância que não possui a IRUP. Para esta instância I, $v(I) = \lceil v_{rl}(I) \rceil + 1$. Tal instância foi construída artificialmente a partir do problema da 4-PARTIÇÃO e apresenta coeficientes da ordem de 10^7, o que levou Marcotte a conjecturar que qualquer instância que não possuísse a IRUP deveria possuir coeficientes da ordem de 10^7, sendo portanto uma instância dissociada de problemas práticos. Fieldhouse [8], no entanto, apresentou uma instância bastante simples que não possui a IRUP: $L = 30, l = \{15, 10, 6\}$ e $d = \{21, 32, 54\}$. Para esta instância I, temos $v_{rl}(I) = 31,9666\ldots$ e $v(I) = 33$.

Scheithauer e Terno [20], Gau [10], Schwerin e Wäscher [22] também exibiram instâncias com coeficientes pequenos, cuja diferença entre $v(I)$ e $v_{rl}(I)$ (*gap*) é maior que 1. Em nossos testes computacionais, também encontramos diversas instâncias cujo *gap* é maior que 1. Schwerin e Wäscher [22] apresentaram uma instância com 200 itens cujo *gap* é 1,14435. Este é o maior *gap* conhecido até o momento. Estes resultados levaram Scheithauer e Terno [20] a definir a *Modified Integer Round-Up Property (MIRUP)*.

Definição 2.2. *Um problema de programação linear inteira P (de minimização) possui a modified integer*

round-up property (MIRUP) se $v(I) \leq \lceil v_{rl}(I) \rceil + 1$ *para toda instância* $I \in P$.

Analogamente, dizemos que uma instância I de um PLI possui a MIRUP se $v(I) \leq \lceil v_{rl}(I) \rceil + 1$. Scheithuaer e Terno [20] provaram que toda instância do PCE_1 (i.e., seus correspondentes PLI) com $m \leq 5$ possuem a MIRUP. Em experimentos computacionais realizados por diversos autores [21, 25], onde foram determinadas soluções inteiras ótimas, verificou-se que todas as instâncias testadas possuíam a MIRUP. Não se conhece, até o momento, nenhuma instância do PCE_1 que não possua a MIRUP. Em 1995, Scheithauer e Terno [21] introduziram a seguinte conjectura.

Conjectura MIRUP: *O* PCE_1 *possui a MIRUP.*

Voltando ao método de resolução descrito no final da seção anterior, chamemos de I uma instância para a qual $\bar{v} = 0$. Esta conjectura afirma que $v(I) \leq \lceil v_{rl}(I) \rceil + 1$. Assim, temos que procurar uma solução inteira com valor $\lceil v_{rl}(I) \rceil$ ou $\lceil v_{rl}(I) \rceil + 1$.

Na próxima seção descrevemos um algoritmo que, dada uma instância do PCE_1 e um número C, encontra uma solução inteira de valor C, se ela existir.

3 O Algoritmo First Fit Decreasing with Backtracking

O algoritmo que desenvolvemos, descrito nesta seção, é fortemente baseado no *First Fit Decreasing (FFD)*, proposto por David Johnson. Trata-se de um algoritmo de aproximação, bastante simples e que apresenta um desempenho bastante satisfatório na prática. Em 1973, Johnson [15] provou que o FFD tem limite de desempenho assintótico $\frac{11}{9}$. Mais precisamente, denotando por $FFD(I)$ o valor da solução encontrada pelo algoritmo FFD e por $OPT(I)$ o valor de uma solução ótima para uma instância I, então para toda instância I, $FFD(I) \leq \frac{11}{9}OPT(I) + 4$. Em 1991, Yue [27] mostrou que a constante aditiva do teorema acima pode ser trocada por 1. Segundo Bramel *et al.* [2], o FFD apresenta desempenho empírico no caso médio de 1,02 e pode ser implementado de forma a ter complexidade de tempo $\mathcal{O}(m \log m)$.

Descrevemos a seguir o algoritmo FFD, adotando as seguintes notações: $\wp_i$ representa o padrão de corte i e $c(\wp_i)$ é uma função que retorna o comprimento da barra não utilizado pelo padrão $\wp_i$.

Algoritmo FFD

Entrada: $(L, l_1, \ldots, l_n)$.
Saída: Uma solução inteira $\wp_1, \ldots, \wp_k$ que atende a demanda.

1 Classifique $l_1, \ldots, l_n$ em ordem decrescente e faça $k = 1$ e $\wp_k = \emptyset$.
2 Para $i = 1$ até n procure $j = min\{h \mid c(\wp_h) \geq l_i \ (h = 1, \ldots, k)\}$
  2.1 Se existir tal j então empacote o item i no padrão $\wp_j$, ou seja, faça $\wp_j = \wp_j \cup \{i\}$.
  2.2 Caso contrário, faça $k = k + 1$ e empacote o item i em $\wp_k$ fazendo $\wp_k = \{i\}$.
3 Retorne $\wp_1, \ldots, \wp_k$ e pare.

Apresentamos a seguir uma variação do algoritmo FFD, que chamamos de *FFD especializado*, denotado por FFD*e*. Na descrição do algoritmo FFD*e*, denotamos por $f(\wp_k)$ uma função que retorna a frequência do item i em $\wp_k$.

Na implementação do FFD*e*, uma preocupação pode ser a quantidade de memória necessária para armazenar a solução. Podemos, no entanto, afirmar que o número de padrões distintos necessários para cortar uma lista S que possui m itens de comprimentos distintos é linear em relação a m, como mostra o seguinte resultado.

Teorema 3.1. *Seja* S *uma lista que possui* m *itens de comprimentos distintos. O algoritmo FFDe encontra uma solução constituída de no máximo* $2m$ *padrões distintos.*

Prova. Por indução em m. Seja $m = 1$. Seja $c = \lfloor \frac{L}{l_1} \rfloor$. Se $c \geq d_1$ a solução encontrada pelo FFD*e* é constituída de apenas um padrão onde o item 1 aparece d_1 vezes. Caso contrário, a solução encontrada pelo FFD*e* utilizará $\lfloor \frac{d_1}{c} \rfloor$ vezes o padrão $\wp_1$, no qual o item 1 aparece c vezes. Seja $r = d_1 - \lfloor \frac{d_1}{c} \rfloor \cdot c$. Se $r > 0$ então será usado mais um padrão, no qual o item 1 aparecerá r vezes.

Considere uma lista com $m \geq 2$ itens. Retirando desta lista o item de menor comprimento, temos uma lista com $m - 1$ itens, e pela hipótese de indução, o FFD*e* aplicado a essa lista encontra uma solução com no máximo $2m - 2$ padrões distintos. A demanda do item de menor comprimento pode, eventualmente, ser atendida utilizando-se o espaço restante nos padrões já gerados. Se isso não for possível, esta demanda pode ser atendida usando-se no máximo mais 2 padrões, como vimos no parágrafo anterior. □

Algoritmo FFDe

Entrada: $(L, l_1, \ldots, l_m, d_1, \ldots, d_m)$.
Saída: Uma solução inteira que atende a demanda.

1 Classifique $l_1, \ldots, l_m$ em ordem decrescente e faça $k = 1$ e $\wp_k = \emptyset$.
2 Repita até que $d_i = 0$ $(i = 1, \ldots, m)$.
  2.1 Procure $j = min\{h \mid l_h \leq c(\wp_k)\ (h = 1, \ldots, m)\}$.
  2.2 Se existir tal j então faça $f = min(d_j, \lfloor \frac{c(\wp_k)}{l_j} \rfloor)$ e empacote f vezes o item j em $\wp_k$, fazendo f vezes $\wp_j = \wp_j \cup \{i\}$.
  2.3 Caso contrário, faça $r_k = min\{\lfloor \frac{d_i}{f_i(\wp_k)} \rfloor,\ i \in \wp_k\}$.
    2.3.1 Para todo $i \in \wp_k$ faça $d_i = d_i - f_i(\wp_k) r_k$. Faça $k = k + 1$ e $\wp_k = \emptyset$.
3 Retorne $\wp_1, \ldots, \wp_k$ e $r_1, \ldots, r_k$ e pare.

Esta versão que propomos é especialmente apropriada por requerer menor tempo de execução e por permitir que a implementação do algoritmo seja feita utilizando-se estruturas de dados mais simples. No entanto, a solução produzida pelo FFDe é a mesma produzida pelo FFD.

Teorema 3.2. *A solução do algoritmo FFDe é equivalente à solução do algoritmo FFD.*

Prova. Por indução em n, o número de itens da lista S. Para $n = 1$, claramente a solução encontrada pelo FFD é igual à solução encontrada pelo FFDe. Considere agora uma lista S com $n \geq 2$ itens ordenados, sem perda de generalidade, em ordem decrescente. Seja $\wp_1, \ldots, \wp_k$ a solução encontrada pelo FFD aplicado à lista S e $\wp_j$ o padrão onde o item n foi empacotado.

Seja S' a lista obtida de S após retirar o item n (que possui o menor comprimento). Claramente a solução encontrada pelo FFD aplicado à lista S' é $\wp_1, \ldots, \wp_j - \{n\}, \ldots, \wp_k$. Pela hipótese de indução, essa solução é igual à solução encontrada pelo FFDe aplicado à S'. Note que o item n cabe em $\wp_j$ e não cabe nos padrões de menor índice.

Uma análise cuidadosa nos permite concluir que ao aplicar o algoritmo FFDe à lista S, o item n também será empacotado em $\wp_j$ pois ao construir os padrões $\wp_1, \ldots, \wp_{j-1}$, o item n não será escolhido (por ser de maior índice). Portanto a solução obtida pelo algoritmo FFDe aplicado a S também será $\wp_1, \ldots, \wp_k$. □

Apesar de ser empiricamente bom no caso médio, Simchi-Levi [23] mostrou que o limite de desempenho absoluto do FFD é 1,5 no pior caso; e que este limite é justo a não ser que $\mathcal{P} = \mathcal{NP}$. Foi também observado que o FFD encontra dificuldade, no que diz respeito à qualidade da solução, em instâncias pequenas ou quando os itens têm comprimento de aproximadamente $\frac{1}{3}L, \frac{1}{4}L, \frac{1}{5}L, \ldots$, onde L é o tamanho da barra (*cf.* Schwerin e Wäscher [22]) ou ainda nas instâncias que Falkenauer [7] chamou de *triplets*. Apresentamos a seguir o algoritmo *First Fit Decreasing with Backtracking (FFDWB)*, baseado no algoritmo FFDe, que encontra uma solução inteira ótima.

Na Seção 1 vimos que o SimplexGC fornece uma solução ótima para a relaxação linear de (1) cujo valor chamamos de v_{rl}. Podemos então usar $v_{ip} = \lceil v_{rl} \rceil$ como um limite inferior para o valor de qualquer solução inteira ótima. Como vimos Seção 2, se a conjectura MIRUP for verdadeira, $v_{ip} + 1$ é um limite superior. O algoritmo FFDWB utiliza estes limites para encontrar mais rapidamente soluções inteiras ótimas para o PCE_1.

A idéia básica do algoritmo FFDWB é aplicar um procedimento semelhante ao algoritmo FFDe para gerar um padrão de cada vez. O desperdício deste padrão é então comparado com o desperdício de uma solução cujo valor é v_{ip}, da forma que explicitaremos mais à frente. Se o padrão gerado não for promissor executamos um retrocesso, diminuindo a frequência dos itens dentro do padrão, do último item que foi empacotado até o primeiro, e aplicamos recursivamente o procedimento na parte da barra não utilizada pelo padrão.

O algoritmo FFDWB, utiliza como subrotina o procedimento PACKING. Este procedimento recebe como parâmetros de entrada um valor D_k, que representa o comprimento que pode ser desperdiçado numa solução que utiliza C barras depois de gerados os padrões $\wp_1, \ldots, \wp_{k-1}$; um valor k, representando o índice do padrão corrente; e ainda os valores f e i, onde f representa quantas vezes o item de comprimento l_i deve ser empacotado no padrão corrente. No procedimento assumimos como constantes globais o valor C, que representa a quantidade máxima de barras que podem ser utilizadas na solução, os comprimentos $l_1, \ldots, l_m$ dos itens e suas demandas $d_1, \ldots, d_m$. O procedimento PACKING com parâmetros D_k, k, f e i, encontra, se existir, uma solução que utiliza no máximo $C - k + 1$ barras onde a primeira barra

tem comprimento $c(\wp_k)$ e as demais barras têm comprimento L. Em [4] mostramos como especializar o algoritmo MOCHILA obtendo o algoritmo MOCHILAe, usado no procedimento PACKING. O algoritmo FFDWB e o procedimento PACKING são descritos a seguir.

Algoritmo FFDWB

Entrada: $(L, l_1, \ldots, l_m, d_1, \ldots, d_m, C)$.
Saída: Uma solução inteira de valor no máximo C, se existir; caso contrário, o valor 0.

1 Classifique os itens em ordem não decrescente de $min(d_i, \lfloor \frac{L}{l_i} \rfloor)$ $(i = 1, \ldots, m)$.
2 Faça $D = CL - \sum_{i=1}^{m} l_i d_i$ e $f = min(d_1, \lfloor \frac{L}{l_1} \rfloor)$.
3 Para $f' = f$ até 0 execute PACKING$(D, 1, f', 1)$.
4 Retorne 0 e pare.

Basicamente, o algoritmo FFDWB enumera todos os padrões viáveis, buscando uma coleção de padrões de cardinalidade no máximo C que atenda toda a demanda. O algoritmo executa uma busca em profundidade numa árvore, onde cada nó da árvore representa um padrão, e a solução encontrada é um ramo da árvore, desde a raiz até uma folha. Esta árvore tem altura no máximo C. Utilizando as desigualdades encontradas no passo 3.4 do procedimento PACKING, podamos a árvore, diminuindo bastante o espaço de busca. Note que, ao percorrer um ramo da árvore com profundidade k, temos que $D_1 \leq \ldots \leq D_k$ (não necessariamente $c(\wp_1) \leq \ldots \leq c(\wp_k)$). Isto significa que o FFDWB prefere gerar inicialmente os padrões que apresentam pequeno desperdício; com isso, os primeiros níveis da árvore possuem poucas ramificações, quando comparados com os níveis posteriores.

Procedimento PACKING(D_k, k, f, i)

1 Empacote f vezes o item i em $\wp_k$, fazendo f vezes $\wp_k = \wp_k \cup \{i\}$, e faça $d_i = d_i - f$.
2 Procure $j = min\{h \mid d_h > 0 \ (h = 1, \ldots, m)\}$. Se não existir tal j retorne $\wp_1, \ldots, \wp_k$ e pare.
3 Procure $i' = min\{h \mid d_h > 0 \ e \ l_h \leq c(\wp_k) \ (h = i+1, \ldots, m)\}$.
  3.1 Se existir i' então faça $f = min(d_{i'}, \lfloor \frac{c(\wp_k)}{l_{i'}} \rfloor)$ e vá para o passo 5.
  3.2 Execute o algoritmo MOCHILAe com parâmetros $L, l_1, \ldots, l_m, l_1, \ldots, l_m, d_1, \ldots, d_m$.
  3.3 Faça $t = L - \sum_{i=1}^{m} l_i z_i$.
  3.4 Se $k < C$ e $c(\wp_k) \leq \lfloor \frac{D_k}{C-k+1} \rfloor$ e $t \leq \lfloor \frac{D_k - c(\wp_k)}{C-k} \rfloor$ então execute
    PACKING$(D_k - c(\wp_k), k+1, min(d_j, \lfloor \frac{L}{l_j} \rfloor), j)$.
4 Faça $f = -1$. {Para que o laço no passo seguinte não seja efetuado}
5 Para $f' = f$ até 0 execute PACKING(D_k, k, f', i'), faça $d_{i'} = d_{i'} + f'$ e remova de $\wp_k$ as f' ocorrências do item i', fazendo f vezes $\wp_k = \wp_k - \{i'\}$.

O seguinte teorema garante a corretude do algoritmo FFDWB.

Teorema 3.3. *O algoritmo FFDWB encontra, se existir, um empacotamento que utiliza no máximo C barras.*

Prova. Seja $\wp_1, \ldots, \wp_C$ um empacotamento que utiliza C barras, ordenadas, sem perda de generalidade, em ordem crescente de desperdício. Seja $D_k = CL - \sum_{i=1}^{m} l_i d_i - \sum_{i=1}^{k-1} c(\wp_i)$. Claramente $c(\wp_k) \leq \lfloor \frac{D_k}{C-k+1} \rfloor$ e $c(\wp_j) \leq \lfloor \frac{D_k - c(\wp_k)}{C-k} \rfloor$ para algum j $(k < j \leq C)$, de outra forma a demanda não poderia ter sido totalmente atendida pelas C barras. Portanto nenhum dos padrões $\wp_1, \ldots, \wp_C$ seria descartado no passo 3.4 do procedimento PACKING. Resta apenas verificar se o procedimento PACKING seria capaz de gerar estes padrões.

Note que cada item, digamos i, aparece em $\wp_1$ no máximo $min(d_i, \lfloor \frac{L - \sum_{j=1}^{i-1} l_j f_j(\wp_1)}{l_i} \rfloor)$ vezes. Conforme o passo 3 do algoritmo FFDWB, a frequência do item 1 irá variar de $min(d_1, \lfloor \frac{L}{l_1} \rfloor)$ até $f_i(\wp_1)$. Conforme o passo 3.1 do procedimento PACKING, a frequência do item 2 irá variar de $min(d_2, \lfloor \frac{L - l_1 f_1(\wp_1)}{l_1} \rfloor)$ até $f_2(\wp_1)$, e assim sucessivamente até que o algoritmo tenha gerado $\wp_1$. De forma análoga construímos $\wp_2, \ldots, \wp_C$, completando a demonstração □

A Figura 1 mostra a árvore percorrida pelo algoritmo FFDWB ao resolver a instância: $L = 20$, $l = \{10, 8, 7, 6, 5, 4\}$, $d = \{3, 1, 1, 4, 1, 1\}$ e $C = 4$. Cada nó da árvore representa um padrão. Observe que os primeiros dois ramos da árvore (olhando de cima para baixo) foram podados no segundo nível, pois os dois primeiros padrões gerados neste nível apresentam desperdício de 2 e 3, enquanto que o passo 3.4 do procedimento PACKING impõe que o segundo padrão apresente desperdício zero $(c(\wp_2) \leq \lfloor \frac{D_2}{C-2+1} \rfloor = 0)$.

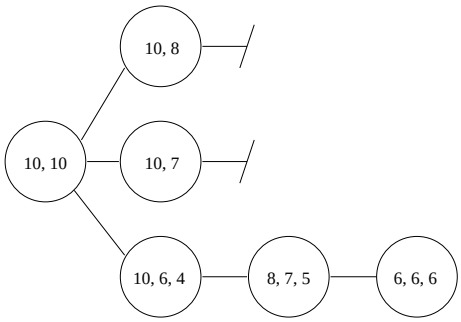

Figura 1: Árvore percorrida pelo algoritmo FFDWB ao resolver a instância caracterizada por $L = 20$, $l = \{10, 8, 7, 6, 5, 4\}$, $d = \{3, 1, 1, 4, 1, 1\}$ e $C = 4$.

A solução encontrada pelo FFDWB é representada pelo terceiro ramo da árvore, e utiliza 4 barras. É interessante notar que o algoritmo FFD aplicado a esta mesma instância encontra uma solução que utiliza 5 barras.

Descrevemos a seguir um algoritmo, que chamamos de algoritmo HÍBRIDO, baseado no método de geração de colunas, que utiliza o FFDWB para resolver o último problema residual. Uma explicação detalhada deste algoritmo pode ser encontrada em [4].

Algoritmo HÍBRIDO

Entrada: $(L, l_1, \ldots, l_m, d_1, \ldots, d_m)$
Saída: Uma solução do problema (1).

1 Faça $v_{rl} = 0$ e $v_{ip} = 0$. (v_{rl} é o valor da solução ótima do problema relaxado; e v_{ip} representa o valor da solução inteira corrente)
2 Execute o algoritmo SimplexGC com parâmetros $L, l_1, \ldots, l_m, d_1, \ldots, d_m$.
Se $\sum_{i=1}^{m} x_i > v_{rl}$ então faça $v_{rl} = \sum_{i=1}^{m} x_i$.
3 Para $i = 1$ até m faça $x_i^* = \lfloor x_i \rfloor$. Faça $v_{ip} = v_{ip} + \sum_{i=1}^{m} x_i^*$
4 Se $x_i^* > 0$ para algum $i = 1, ..., m$ então
  4.1 Retorne A e $x_1^*, \ldots, x_m^*$. Faça $m' = 0$.
  4.2 Para $i = 1$ até m faça
    4.2.1 Para $j = 1$ até m faça $d_i = d_i - a_{ij} x_j^*$.
  4.3 Para $i = 1$ até m faça
    4.3.1 Se $d_i > 0$ faça $m' = m' + 1$, $l_{m'} = l_i$ e $d_{m'} = d_i$.
  4.4 Se $m' = 0$ então pare.
  4.5 Faça $m = m'$ e retorne ao passo 2.
5 Faça $C = \lceil v_{rl} \rceil - v_{ip}$. Execute o FFDWB com parâmetros $L, l_1, \ldots, l_m, d_1, \ldots, d_m, C$.
6 Se o FFDWB não retornou 0 então retorne $\wp_1, \ldots, \wp_C$ e pare.
7 Caso contrário, execute o FFDWB com parâmetros $L, l_1, \ldots, l_m, d_1, \ldots, d_m, C + 1$.
8 Se o FFDWB não retornou 0 então retorne $\wp_1, \ldots, \wp_{C+1}$ e pare.
9 Caso contrário, execute o FFD*e* com parâmetros $L, l_1, \ldots, l_m, d_1, \ldots, d_m$,
retorne a solução encontrada e pare.

A conjectura MIRUP explica porque no algoritmo HÍBRIDO não usamos o FFDWB para buscar soluções com $C + 2, C + 3, \ldots$ barras. Se, no passo 8 do algoritmo HÍBRIDO, o FFDWB não encontrar uma solução que utiliza $C + 1$ barras, então o problema residual correspondente constitui uma instância que não possui a MIRUP, e então teremos refutado a conjectura MIRUP.

Sob a hipótese de que a conjectura MIRUP é verdadeira, temos o seguinte resultado, que estima a qualidade da solução encontrada pelo algoritmo híbrido.

Teorema 3.4. *Se a conjectura MIRUP for verdadeira, a diferença entre o valor da solução encontrada pelo algoritmo HÍBRIDO e o valor de uma solução inteira ótima é no máximo 1.*

Prova. O valor $\lceil v_{rl} \rceil$ é, claramente, um limite inferior para o valor de uma solução inteira ótima. Assumindo ser verdadeira a conjectura MIRUP, o algoritmo jamais chegará ao passo 9, pois o problema residual a ser resolvido no passo 8 tem solução cujo valor não excede C, portanto, pela conjectura MIRUP, possui solução inteira cujo valor é no máximo $C+1$, assim o algoritmo irá parar no passo 7 ou no passo 8.□

4 Resultados Computacionais

Avaliamos o algoritmo híbrido delineado na seção anterior resolvendo 4000 instâncias geradas aleatoriamente e mais cerca de duas centenas de instâncias práticas tiradas das plantas de ferragem de obras de uma construtora. O algoritmo híbrido foi implementado usando-se a linguagem C e os testes foram executados numa estação Sun Sparc 1000, clock de 50 mhz e 704 MB de memória principal. Utilizamos o CPLEX 2.0 [5] para resolver os sistemas de equações lineares que aparecem nos passos 3 e 6 do algoritmo SimplexGC.

4.1 Instâncias Aleatórias

Resolvemos 4000 instâncias geradas aleatoriamente utilizando o método delineado por Wäscher e Gau em [26] e que denotaremos por *instâncias de W & G*. Tais instâncias foram geradas usando o algoritmo *CUTGEN1*, descrito em [12]. A Tabela 1 apresenta os resultados da aplicação do algoritmo HÍBRIDO às instâncias de W & G. Nela estão indicados o número de itens (m), o tamanho dos itens em relação à barra, o número de instâncias para as quais foi encontrada uma solução que satisfaz a IRUP (portanto, uma solução ótima), o número de instâncias para as quais foi encontrada uma solução que satisfaz a MIRUP mas não satisfaz a IRUP, o número médio de colunas geradas pelo SimplexGC em cada instância, o tempo médio requerido para resolver cada instância, o ganho médio percentual da solução encontrada pelo algoritmo HÍBRIDO em relação à solução encontrada pelo FFD e finalmente o desperdício médio em cada instância.

m	Tamanho dos itens	IRUP	MIRUP	Colunas geradas	Tempo (seg)	Ganho sobre FFD	Desperdício
10	0%-100%	200	0	7,21	0,04	0,345%	13,734%
	0%-75%	200	0	11,92	0,07	0,569%	15,084%
	0%-50%	200	0	23,77	0,14	2,727%	3,091%
	0%-25%	200	0	41,02	0,34	1,590%	1,388%
20	0%-100%	199	1	26,01	0,17	0,273%	9,873%
	0%-75%	199	1	47,37	0,32	0,641%	7,450%
	0%-50%	200	0	108,98	0,97	2,322%	0,695%
	0%-25%	200	0	105,60	1,24	0,864%	0,665%
30	0%-100%	199	1	56,70	0,45	0,225%	10,046%
	0%-75%	200	0	112,53	1,02	0,545%	6,376%
	0%-50%	198	2	293,29	3,87	1,887%	0,362%
	0%-25%	200	0	187,85	2,71	0,551%	0,451%
40	0%-100%	200	0	99,31	0,85	0,200%	9,901%
	0%-75%	198	2	226,31	2,66	0,633%	4,301%
	0%-50%	200	0	585,84	9,50	1,456%	0,206%
	0%-25%	200	0	289,55	4,14	0,435%	0,348%
50	0%-100%	200	0	159,15	1,55	0,227%	7,175%
	0%-75%	197	3	375,12	4,78	0,582%	4,399%
	0%-50%	198	2	969,48	20,28	1,157%	0,157%
	0%-25%	200	0	397,06	17,80	0,363%	0,275%

Tabela 1: O Algoritmo HÍBRIDO aplicado às instâncias de W & G.

Foram encontradas soluções inteiras ótimas para pelo menos 3988 instâncias e o tempo médio necessário para resolver cada instância foi inferior a 4 segundos.

A Tabela 2 compara os resultados obtidos pelo algoritmo HÍBRIDO e diversos métodos encontrados na literatura, quando aplicados às instâncias de W & G. Adotamos para estes métodos os nomes sugeridos por Wäscher e Gau [26] e indicamos as referências onde o leitor pode obter detalhes deste métodos. Dentre todos eles, o algoritmo HÍBRIDO foi o que encontrou soluções ótimas para o maior número de instâncias. É pertinente observar que os resultados obtidos por Wäscher e Gau em 1996 eram os melhores até então.

Os resultados dos testes com essas 4000 instâncias aleatórias vêm fortalecer a conjectura MIRUP. Além destas instâncias, resolvemos também *alguns milhões* de instâncias aleatórias, com m variando entre 10

Algoritmo	$m = 10$	$m = 20$	$m = 30$	$m = 40$	$m = 50$	Total
Total de instâncias	800	800	800	800	800	4000
HÍBRIDO	800	798	797	798	795	3988
RSUC [10]	797	792	790	779	763	3921
ROPT [26]	796	788	782	770	738	3874
CSTAOPT [24]	758	754	731	732	734	3709
RFFD [26]	779	758	743	726	695	3701
CSTAFFD [26]	752	746	715	716	701	3630
FFD	469	378	359	321	318	1845
BOPT [26]	428	321	262	251	210	1472
BRURED [19]	325	193	129	94	80	821
BRUSUC [24]	257	142	74	53	52	578
BRUSIM [26]	93	47	23	15	10	188

Tabela 2: Soluções inteiras ótimas obtidas por diversos algoritmos aplicados às instâncias de W & G.

e 20, com o objetivo de encontrar um contra-exemplo para a conjectura MIRUP. Não encontramos uma instância sequer com *gap* entre v_{ip} e v_{rl} maior que 1,14435.

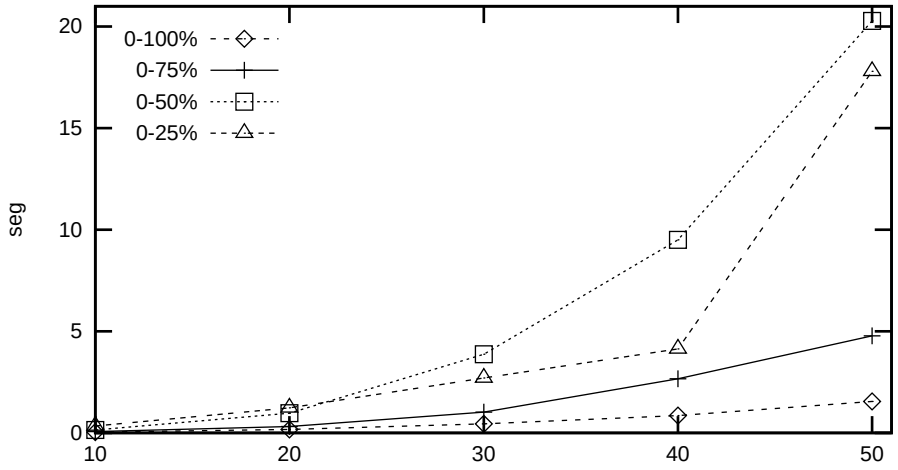

Figura 2: Tempo médio requerido para resolver as instâncias de W & G.

4.2 Instâncias Reais

Apresentamos nesta subseção os resultados obtidos ao aplicar o algoritmo HÍBRIDO a cerca de duas centenas de instâncias reais tiradas das plantas de ferragem de duas obras de uma construtora com atuação em várias cidades do país. Tratam-se de edifícios gêmeos de 9 andares, construídos lado a lado. As instâncias consideradas são relativas ao problema de determinar como cortar o aço a ser utilizado nas estruturas de concreto armado de modo a minimizar o desperdício de aço. Nestas obras foi utilizado aço CA-60B com 5mm de bitola, e aço CA-50A com bitolas 6.3mm, 8mm, 10mm, 12.5mm, 16mm e 20mm. Dessa forma houve necessidade de utilizar 7 tipos de barras de aço, originando 7 categorias de problemas. A partir das 42 plantas de ferragem analisadas, obtivemos 209 instâncias, que chamaremos de *instâncias práticas*.

A Tabela 3 mostra os resultados obtidos ao resolver as instâncias práticas usando o algoritmo HÍBRIDO. Foram encontradas, dentro de um tempo bastante curto, soluções ótimas para todas as instâncias. Ademais, o algoritmo HÍBRIDO obteve um ganho considerável em relação ao FFD.

No entanto, apesar de termos encontrado soluções ótimas para todas as instâncias, em algumas categorias o desperdício médio foi bastante alto. Com o intuito de diminuir o desperdício, agrupamos todas as instâncias de cada categoria gerando apenas 7 instâncias, que chamaremos de *instâncias agrupadas*. Os resultados obtidos pelo algoritmo HÍBRIDO ao resolver as instâncias agrupadas são apresentados na Tabela 4.

Bitola	Nº de problemas	m médio	Tamanho dos itens	IRUP	Colunas geradas	Tempo (seg)	Ganho sobre FFD	Desperdício
5.0	13	7	2%-50%	13	14,54	0,23	0,945%	0,324%
6.3	43	11	2%-98%	43	23,86	0,33	1,868%	1,438%
8.0	40	12	2%-90%	40	30,10	0,35	1,128%	6,281%
10.0	35	20	4%-92%	35	78,00	1,03	0,825%	7,843%
12.5	36	14	6%-92%	36	46,03	0,64	2,207%	5,467%
16.0	26	13	16%-75%	26	25,12	0,31	1,353%	20,550%
20.0	16	8	20%-74%	16	9,69	0,12	0,894%	12,023%

Tabela 3: Resultados obtidos pelo algoritmo HÍBRIDO aplicado às instâncias práticas.

Bitola	m	Tamanho dos itens	IRUP	Colunas geradas	Tempo (seg)	Ganho sobre FFD	Desperdício
5.0	48	2%-50%	Sim	118	41	0,607%	0,020%
6.3	209	2%-98%	Sim	622	37	0,474%	0,017%
8.0	264	2%-90%	Sim	2706	190	1,278%	0,009%
10.0	365	4%-92%	Sim	10758	1576	1,365%	0,061%
12.5	301	6%-92%	Sim	8333	782	1,481%	0,268%
16.0	218	16%-75%	Sim	1231	164	1,673%	4,995%
20.0	105	20%-74%	Sim	573	29	0,489%	2,507%

Tabela 4: Resultados obtidos pelo algoritmo HÍBRIDO aplicado às instâncias agrupadas.

Para se ter uma idéia mais aproximada do economia, precisamos levar em conta o peso das barras de aço. A Tabela 5 apresenta a quantidade de aço especificada nas plantas e as quantidades utilizadas nas soluções apresentadas nas tabelas 3 e 4. Vale salientar que esta construtora desperdiça em média, em obras deste tipo, algo em torno de 10% do aço, e este percentual é bom se comparado ao desperdício médio verificado na indústria de construção civil nacional. Agrupando as instâncias, reduzimos o desperdício de aço a cerca de 1%, o que representa uma economia considerável.

		Quantidade	Tabela 5		Tabela 6	
Bitola	Peso (kg/m)	de aço nas plantas (kg)	Quantidade de aço (kg)	Desperdício	Quantidade de aço (kg)	Desperdício
5.0	0,16	5062,008	5078,400	0,324%	5063,040	0,020%
6.3	0,25	15834,380	16062,000	1,438%	15837,000	0,017%
8.0	0,40	24415,352	25948,800	6,281%	24417,600	0,009%
10.0	0,63	23799,497	25666,200	7,843%	23814,000	0,061%
12.5	1,00	24235,060	25560,000	5,467%	24300,000	0,268%
16.0	1,60	15305,872	18470,400	20,675%	16070,400	4,995%
20.0	2,50	17969,525	20130,000	12,023%	18420,000	2,507%
Totais		126621,695	136915,800	8,130%	127922,040	1,027%

Tabela 5: Desperdício de aço nas soluções encontradas para as instâncias práticas.

Considerações Finais

Discorremos sobre resultados teóricos [20] e práticos [21, 25] que sugerem ser verdadeira a conjectura MIRUP. Os testes que realizamos com instâncias aleatórias e práticas vieram a fortalecer esta conjectura. Esta conjectura nos fornece um excelente limitante para o valor de uma solução inteira ótima. Utilizando este limitante, propusemos o algoritmo exato FFDWB, que mostrou-se eficiente ao resolver instâncias pequenas do PCE_1.

Mostramos como utilizar o FFDWB como subrotina no algoritmo HÍBRIDO, de modo a resolver

instâncias maiores do PCE_1. O algoritmo HÍBRIDO reúne algoritmos de diversas naturezas, como o SimplexGC, o FFDWB e o FFD*e*. Deste fato deriva o seu nome. Esta abordagem diversificada tem se mostrado promissora, e tem sido explorada em trabalhos recentes[24, 21, 26, 11]. Mostramos também que a diferença entre o valor da solução encontrada pelo algoritmo HÍBRIDO e o valor de uma solução inteira ótima é no máximo 1, se verdadeira a conjectura MIRUP. Dizemos então que ele é um algoritmo *quase-exato*.

Repetimos os testes realizados por Wäscher e Gau [26] com 4000 instâncias aleatórias e constatamos que o algoritmo HÍBRIDO obteve desempenho superior ao de todos os algoritmos testados por estes autores. Encontramos uma solução inteira ótima para pelo menos 99,7% dessas instâncias e o tempo médio requerido para resolver cada uma das 4000 instâncias foi inferior a 4 segundos.

Resolvemos também 216 instâncias práticas, incluindo uma instância com 365 itens, sendo tendo encontrado soluções ótimas para todas essas instâncias. Os resultados obtidos, tanto nos testes realizados com instâncias geradas aleatoriamente, quanto nos testes com instâncias práticas, demonstraram a eficiência do algoritmo HÍBRIDO em termos de qualidade da solução e tempo de execução.

O algoritmo HÍBRIDO difere dos algoritmos RSUC e ROPT, propostos por Gau e Wäescher apenas na forma como resolvem o último problema residual. O algoritmo HÍBRIDO utiliza o FFDWB enquanto que o RSUC e o ROPT usam o MTP proposto por Martello e Toth [18]. Seria interessante comparar diretamente estes dois algoritmos, FFDWB e MTP, para entender porque o FFDWB apresenta melhores resultados.

Referências

[1] S. Baum and L. E. T. Jr., *Integer rounding for polymatroid and branching optimization problems*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 416–425.

[2] J. Bramel, W. T. Rhee, and D. Simchi-Levi, *Average-case analysis of the bin-packing problem with general cost structures*, Naval Res. Logist., 44 (1997), pp. 673–686.

[3] V. Chvátal, *Linear Programming*, W. H. Freeman and Company, New York, 1980.

[4] G. F. Cintra and Y. Wakabayashi, *Um algoritmo híbrido para o problema de corte unidimensional*, in XXX Simpósio Brasileiro de Pesquisa Operacional, Curitiba, 1998.

[5] CPLEX, *Using the CPLEX Callable Library and CPLEX Mixed Integer Library*, CPLEX Optimization, Inc, 1995.

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

[7] E. Falkenauer, *A hybrid grouping genetic algorithm for bin packing*, Journal of Heuristics, 2 (1996), pp. 5–30.

[8] M. Fieldhouse, *The duality gap in trim problems*, SICUP-bulletin, 5 (1990).

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*, W. H. Freeman and Co., San Fransisco, 1979.

[10] T. Gau, *Quasi-exact and heuristic algorithms for the standard one-dimensional cutting stock problem*, tech. report, Technische Universitaet Braunschweig, 1994.

[11] ——, *Solution methods for the standard one-dimensional cutting stock problem*, Physica, Heidelberg, 1997.

[12] T. Gau and G. Wäsher, *CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem*, European Journal of Operations Research, 84 (1995), pp. 572–579.

[13] P. Gilmore and R. Gomory, *A linear programming approach to the cutting stock problem*, Operations Research, 9 (1961), pp. 849–859.

[14] ——, *A linear programming approach to the cutting stock problem - part II*, Operations Research, 11 (1963), pp. 863–888.

[15] D. S. Johnson, *Near-optimal bin packing algorithms*, PhD thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1973.

[16] O. Marcotte, *The cutting stock problem and integer rounding*, Mathematical Programming, 33 (1985), pp. 82–92.

[17] ——, *An instance of the cutting stock problem for which the rounding property does not hold*, Oper. Res. Lett., 4 (1986), pp. 239–243.

[18] S. Martello and P. Toth, *Knapsack problems*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Ltd., Chichester, 1990. Algorithms and computer implementations.

[19] K. Neumann and M. Morlock, *Operations Research*, Carl Hanser Verlag, Munich, 1993.

[20] G. Scheithauer and J. Terno, *About the gap between the optimal values of the integer and continuous relaxation one-dimensional cutting stock problem*, in Operations Research Proceedings, Berlin, 1992, Springer-Verlag.

[21] ——, *The modified integer round-up property of the one-dimensional cutting stock problem*, European Journal of Operations Research, 84 (1995), pp. 562–571.

[22] P. Schwerin and G. Wäscher, *The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP*, International Transactions in Operational Research, 4 (1997), pp. 377–389.

[23] D. Simchi-Levi, *New worst-case results for the bin-packing problem*, Naval Res. Logist., 41 (1994), pp. 579–585.

[24] H. Stadtler, *A one-dimensional cutting stock problem in the aluminium industry and its solution*, European Journal of Operations Research, 44 (1990).

[25] G. Wäscher and T. Gau, *Two approaches to the cutting stock problem*, in IFORS'93 Conference, Lisbon, 1993.

[26] ——, *Heuristics for the integer one-dimensional cutting stock problem: a computational study*, OR Spektrum, 18 (1996), pp. 131–144.

[27] M. Yue, *A simple proof of the inequality* $\mathrm{FFD}(L) \leq \frac{11}{9}\mathrm{OPT}(L) + 1$, $\forall L$ *for the FFD bin-packing algorithm*, Acta Math. Appl. Sin., Engl. Ser., 4 (1991), pp. 321–331.