Um Algoritmo Híbrido para o Problema de Corte Unidimensional[1]

Glauber Cintra e Yoshiko Wakabayashi

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, yw@ime.usp.br

Investigamos o problema de corte (de estoque) unidimensional, formulando-o como um problema de programação linear inteira, e propomos um algoritmo híbrido, baseado no método de geração de colunas e num algoritmo exato. O algoritmo exato que desenvolvemos é adequado para resolver instâncias pequenas do problema de corte unidimensional quando se conhece previamente um limitante inferior para o valor da solução inteira ótima. Mostramos ainda que o algoritmo híbrido proposto encontra uma solução inteira cujo valor objetivo difere do valor objetivo ótimo de no máximo 1, se a conjectura MIRUP (Modified Integer Round-Up Property) for verdadeira. Os resultados obtidos na resolução de um expressivo número de instâncias práticas e instâncias geradas aleatoriamente são analisados, indicando um desempenho bastante satisfatório do algoritmo híbrido.

Palavras-chave: problemas de corte e empacotamento, corte unidimensional, geração de colunas.

1 Introdução

O *problema de corte (de estoque) unidimensional* (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 menor número de barras necessário para atender a demanda, ou seja, produzir d_i itens, cada qual de comprimento l_i. Obviamente também estamos interessados em determinar como as barras devem ser cortadas.

O PCE_1 é um problema $\mathcal{NP}$-difícil, amplamente investigado desde a década de 70, para o qual se conhecem vários algoritmos de aproximação, algoritmos exatos e heurísticas [27, 16, 8, 1, 14, 15, 17].

Propomos um algoritmo híbrido para resolver o PCE_1, baseado no método de geração de colunas e num algoritmo exato. Este algoritmo híbrido consiste basicamente em resolver a relaxação linear do problema de programação linear inteira correspondente ao PCE_1 e arredondar para baixo as variáveis fracionárias gerando-se, eventualmente, um problema residual. A relaxação linear é resolvida usando-se o algoritmo Simplex, no qual, a partir de uma solução básica inicial (que pode ser facilmente obtida), a cada iteração uma nova coluna é gerada resolvendo-se um problema da mochila. Este procedimento é repetido até que as variáveis associadas à solução da relaxação linear arrendondadas para baixo sejam todas iguais a zero. Este último problema residual é então resolvido com o algoritmo exato que propomos, denominado *First Fit Decreasing With Backtracking* (FFDWB). O FFDWB é adequado para resolver instâncias pequenas do PCE_1 quando se conhece previamente um limite inferior para o valor da solução inteira ótima.

Basicamente, o algoritmo FFDWB 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. Utilizando-se o limitante previamente conhecido podemos, ao gerar soluções parciais, estimar o desperdício final que será obtido, o que nos permite podar a árvore, diminuindo bastante o espaço de busca.

Mostramos ainda que o algoritmo híbrido proposto encontra uma solução inteira cujo valor objetivo difere do valor objetivo ótimo de no máximo 1, se a conjectura MIRUP (Modified Integer

[1]Pesquisa desenvolvida dentro do Projeto PCE (ProTeM-CC-III/CNPq, Proc. 680082/95-6), Projeto FAPESP (Proc. 96/04505-2) e ProNEx 107/97 (MCT/FINEP); bolsa individual de pesquisa do CNPq (yw/Proc. 304527/89-0).

Round-Up Property) for verdadeira. Por esta conjectura, o valor objetivo de uma solução inteira ótima do PCE_1 é menor ou igual ao valor objetivo de uma solução ótima arredondada para cima mais 1. Dizemos então que o algoritmo híbrido é assintoticamente exato, se verdadeira a conjectura MIRUP.

Os resultados obtidos na resolução de um expressivo número de instâncias práticas e instâncias geradas aleatoriamente são analisados, indicando um desempenho bastante satisfatório do algoritmo híbrido. Resolvemos 4000 instâncias geradas aleatoriamente, agrupadas em 40 classes com características distintas e verificamos que o algoritmo híbrido apresentou desempenho superior a diversos métodos encontrados na literatura que foram aplicadas a estas mesmas 4000 instâncias [29].

Resolvemos ainda cerca de duas centenas de instâncias práticas tiradas das plantas de ferragem de obras de uma construtora e verificamos que o algoritmo híbrido encontrou uma solução ótima para todas estas instâncias. O desperdício total de aço nestas obras reduziu-se para aproximadamente 1%.

Este artigo está organizado da seguinte maneira. Na Seção 2 mostramos como o PCE_1 pode ser formulado como um problema de Programação Linear Inteira (PLI). Na Seção 3 discutimos o método de geração de colunas, usado para resolver a relaxação linear do problema linear inteiro correspondente ao PCE_1. Na Seção 4 descrevemos como, a partir de uma solução fracionária do problema relaxado, podemos obter uma solução inteira sem que haja excesso de produção, e que se aproxime da solução inteira ótima. Nessa seção apresentamos três algoritmos, o *First Fit Decreasing* (FFD), o *First Fit Decreasing especializado* (FFDe) e o *First Fit Decreasing With Backtracking* (FFDWB). O segundo é uma especialização do primeiro, e o terceiro é baseado no segundo. Na Seção 5 descrevemos o algoritmo HÍBRIDO que desenvolvemos. Na Seção 6, mencionamos a conjectura MIRUP (*Modified Integer Round-Up Property*) e a sua relação com o algoritmo HÍBRIDO. Na Seção 7 apresentamos os resultados computacionais obtidos. Para finalizar, na última seção tecemos algumas considerações a respeito deste trabalho.

2 Formulação do Problema

O PCE_1 pode ser formulado como um problema de Programação Linear Inteira (PLI). Para fazer isso, consideramos cada possível forma de cortar uma barra, que chamamos de *padrão de corte* (ou simplesmente *padrão*), e o representamos 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. Por exemplo, no caso de uma instância que consiste de uma barra com comprimento $L = 30$ e 3 itens de comprimentos $l_1 = 5$, $l_2 = 12$, $l_3 = 7$ com demandas $d_1 = 4$, $d_2 = 2$, $d_3 = 2$, respectivamente, podemos ter um padrão j onde $a_j = [3, 0, 2]$. Esse vetor a_j indica que o item 1 ocorre 3 vezes, o item 2 não ocorre nenhuma vez, e o item 3 ocorre 2 vezes no padrão j.

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 padrões viáveis e decidir quantas vezes cada padrão deve ser utilizado de modo a atender a demanda, minimizando o número total de barras utilizadas.

Para isso, introduzimos uma variável x cujos elementos são inteiros $x_j\,(j = 1, \ldots, n)$, que indicam quantas vezes o padrão j é selecionado. Note que se x é uma solução viável do problema acima, então o valor $\sum_{j=1}^{n} x_j$ corresponde ao número de barras a serem cortadas, já que cada padrão corresponde a uma barra. Assim, denotando por A a matriz $m \times n$ cujas colunas são os vetores $a_1, \ldots, a_n$, e representando por d o vetor das demandas, o problema pode ser assim formulado:

$$\min \sum_{j=1}^{n} x_j$$

$$\begin{aligned} &\text{sujeito a } Ax = d \\ &\qquad x_j \geq 0 \text{ e inteiro} \qquad j = 1, \ldots, n. \end{aligned} \tag{1}$$

A formulação acima 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 duas dificuldades, Gilmore e Gomory propuseram o método de geração de colunas [12, 13], delineado na seção seguinte.

3 Geração de Colunas

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 nova matriz A, que chamaremos de *base*, resolvemos a relaxação linear de (1):

$$\begin{aligned} &\min \textstyle\sum_{j=1}^{n} x_j \\ &\text{sujeito a } Ax = d \\ &\qquad x_j \geq 0 \qquad j = 1, \ldots, n. \end{aligned} \tag{2}$$

O método de geração de colunas consiste basicamente em, a partir de uma solução básica viável inicial, representada pela matriz A, usar o algoritmo *simplex revisado* [4], 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. As soluções encontradas por este método convergem para uma solução ótima, não necessariamente inteira. Para contornar este problema, Gilmore e Gomory [12] propuseram arredondar para cima o valor das variáveis, após achar a 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.

Para resolver (2), primeiramente precisamos determinar a matriz A. Felizmente, é fácil obter uma matriz A que permita calcular trivialmente uma solução básica viável para o problema de corte linear. Definimos, no passo 1, a matriz A (diagonal), de dimensão $m \times m$, fazendo $a_{ij} = \lfloor \frac{L}{l_i} \rfloor$ se $i = j$; e $a_{ij} = 0$, caso contrário ($i = 1, \ldots, m; j = 1, \ldots, m$). No passo 2, calculamos uma solução x inicial fazendo $x_i = \frac{d_i}{a_{ii}}$ ($i = 1, \ldots, m$).

Cada iteração do método consiste em tentar gerar uma nova coluna que deve entrar na base. Se tal coluna não existir estamos então na solução ótima. Caso contrário, encontramos uma coluna que deve sair, substituindo-a pela nova coluna gerada. Para encontrar uma nova coluna que deve entrar na base precisamos primeiramente, no passo 3, resolver $yA = c$, onde c é o vetor-linha com entradas $c_i = 1$ ($i = 1, \ldots, m$). Usando a variável y, geramos, no passo 4, uma nova coluna resolvendo o seguinte problema da mochila:

$$\begin{aligned} &\max \textstyle\sum_{i=1}^{m} y_i z_i \\ &\text{sujeito a } \textstyle\sum_{i=1}^{m} l_i z_i \leq L \\ &\qquad z_i \geq 0 \text{ e inteiro} \qquad i = 1, \ldots, m\ . \end{aligned} \tag{3}$$

Note que se z é uma solução viável do problema (3), então z corresponde a um padrão viável. Daqui por diante chamaremos de *valor* de uma solução o valor da função objetivo associado à essa solução. No passo 5, verificamos se o valor da solução ótima de (3) é menor ou igual a 1. Em caso afirmativo, a solução x corrente é uma solução ótima de (2). Nesse caso o algoritmo retorna x e pára. Caso contrário, o vetor z deve entrar na base. Para encontrar uma coluna que deve sair da base primeiramente calculamos, no passo 6, o vetor coluna b, tal que $Ab = z$. No passo seguinte calculamos $t = min\{\frac{x_i}{b_i} \mid b_i > 0\ (i = 1, \ldots, m)\}$. Deve sair da base uma coluna a_s tal que:

$$s = min\{i \mid \frac{x_i}{b_i} = t\ (i = 1, \ldots, m)\}. \tag{4}$$

Calculamos s e, no passo 8, substituímos a_s por z, obtendo a nova base, e calculamos a nova solução corrente x fazendo $x_i = x_i - b_i t$, se $i \neq s$; e $x_i = t$, caso contrário $(i = 1, \ldots, m)$. No passo 9 a iteração é finalizada retornando ao passo 3.

Descrevemos a seguir um algoritmo bastante simples, mas satisfatório para os nossos propósitos, que resolve o problema da mochila (3).

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 Ordene 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 $k = max\{i \mid z_i > 0\ e\ \sum_{j=1}^i y_j \bar{z}_j + \frac{y_{i+1}}{l_{i+1}}(L - \sum_{j=1}^i l_j \bar{z}_j) > M\ (i = m-1, \ldots, 1)\}$.
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.

Apresentamos a seguir o algoritmo SimplexGC.

Algoritmo SimplexGC

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

$$\begin{aligned} \min\ & \textstyle\sum_{j=1}^n x_j \\ \text{sujeito a}\ & Ax = d \\ & x_j \geq 0 \qquad j = 1, \ldots, n, \end{aligned}$$

onde cada coluna j de A é tal que $\sum_{i=1}^m a_{ij} l_i \leq L$.

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.

4 Obtendo uma Solução Inteira

Aplicando o método de geração de colunas 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, uma parte da demanda não foi atendida, pois $\bar{x}_j < x_j$ para algum $j \in \{1, \ldots, m\}$ e portanto $\sum_{j=1}^{m} a_{ij}\bar{x}_j < d_i$ para tal j. 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 o problema residual utilizando um algoritmo que forneça uma solução inteira, como o *First Fit Decreasing (FFD)*, o *First Fit Decreasing especializado (FFDe)* e o *FFDWB*, abordados nas próximas subseções.

4.1 O Algoritmo FFD

Escolhemos utilizar o FFD pois este algoritmo é bem simples, e além disso, apresenta um desempenho bastante satisfatório. Em 1973, Johnson [14] provou que o algoritmo 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, $FFD(I) \leq \frac{11}{9}OPT(I) + 4$. Ademais, o FFD apresenta desempenho empírico no caso médio de 1,02 (*cf.* Bramel *et al.* [3]) e pode ser implementado de forma a ter complexidade de tempo $\mathcal{O}(n \log n)$. 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 $\wp_i$.

A entrada do algoritmo consiste do comprimento L de uma barra e de uma lista S de n itens de comprimentos $l_1, \ldots, l_n$. Neste caso, estamos supondo que a entrada correspondente ao problema residual, que consiste de m itens de comprimento l_i $(i = 1, \ldots, m)$ e demanda d_i, é transformada numa entrada para o algoritmo FFD criando-se uma lista S onde cada item de comprimento l_i aparece d_i vezes.

Algoritmo FFD

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

1 Ordene $l_1, \ldots, l_n$ 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.

Primeiramente, ordenamos os itens em ordem decrescente de comprimento. Iniciamos então o processo empacotando o item 1 no padrão $\wp_1$ e fazendo $k = 1$. Suponha que o item i é o próximo item a ser empacotado. Procuramos dentre os padrões $\wp_1, \ldots, \wp_k$ aquele de menor índice onde é possível empacotar o item i. Caso não exista tal padrão, iniciamos a geração de um novo padrão, ou seja, fazemos $k = k + 1$ e empacotamos o item i no padrão $\wp_k$. Este processo é repetido até que todos os itens tenham sido empacotados, após o que retornamos a solução $\wp_1, \ldots, \wp_k$.

4.2 O Algoritmo FFD especializado (FFDe)

Apresentaremos a seguir uma variação do algoritmo FFD, que chamaremos de *FFD especializado*, denotado por FFDe, especializado para o problema que queremos resolver. Tal versão 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.

Seja S uma lista de m itens de comprimentos $l_1, \ldots, l_m$ e demandas $d_1, \ldots, d_m$. Ordene, no passo 1, os elementos de S em ordem decrescente de comprimento e faça $k = 1$. No passo seguinte, repetimos o procedimento descrito no próximo parágrafo até que todos os itens sejam empacotados, ou seja, até obtermos $d_i = 0$ $(i = 1, \ldots, m)$, depois do que o algoritmo retorna $\wp_1, \ldots, \wp_k$ e $r_1, \ldots, r_k$.

Procuramos o item de menor índice que caiba em $\wp_k$. Se existir tal item, empacotamos este item no padrão $\wp_k$ o maior número de vezes possível, limitado à demanda do item, e atualizamos a demanda. Caso contrário, determinamos r_k, que é o número de vezes que o padrão $\wp_k$ deve ser utilizado, fazendo $r_k = min(\lfloor \frac{d_i}{f_i(\wp_k)} \rfloor,\ i \in \wp_k)$, onde $f_i(\wp_k)$ é uma função que retorna a frequência do item i em $\wp_k$, atualizamos as demandas fazendo $d_i = d_i - f_i(\wp_k) r_k$ para todo item i contido em $\wp_k$ e fazemos $k = k + 1$.

Algoritmo FFDe

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

Note que, dado um item i, o algoritmo FFD procura imediatamente determinar em qual padrão este item deve ser cortado. Já no algoritmo FFDe, dado um padrão k, procuramos qual o próximo item que deve ser empacotado nesse padrão. Chamamos o algoritmo FFD de *item-orientado* e o algoritmo FFDe de *padrão-orientado*. Esta característica, ser um algoritmo padrão-orientado, permite adaptar, sem maiores dificuldades, o FFDe obtendo o algoritmo FFDWB, que será detalhado na próxima subseção. Apesar deste enfoque diferenciado entre o FFD e o FFDe, vale o seguinte resultado.

Teorema 4.1. *Dada uma lista S de itens, a solução do algoritmo FFD é igual à solução do algoritmo FFDe.*

Apesar de ser empiricamente bom no caso médio, Simchi-Levi [25] 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 onde 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 [24]) ou ainda nas instâncias que Falkenauer [6] chamou de *triplets*. Apresentamos a seguir o algoritmo FFDWB, baseado no algoritmo FFDe, que encontra uma solução inteira *ótima*.

4.3 O Algoritmo FFDWB

Na Seção 3 vimos que o método de geração de colunas fornece uma solução ótima para a relaxação linear de (1) cujo valor chamaremos 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 veremos na Seção 6, este limitante é justo na grande maioria dos problemas.

A idéia básica do algoritmo *First Fit Decreasing with Backtracking (FFDWB)*, que propomos, 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 da 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 cortado até o primeiro, e aplicamos recursivamente o procedimento semelhante ao FFDe na parte da barra não utilizada pelo padrão. Na solução encontrada pelo FFDWB, cada padrão é utilizado uma vez, de modo que a cada padrão corresponde uma barra.

Para facilitar a compreensão do algoritmo FFDWB, detalhamos primeiramente 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 $d_1, \ldots, d_m$.

No passo 1, empacotamos f vezes o item i em $\wp_k$ e atualizamos a demanda do item i fazendo $d_i = d_i - f$. Procuramos então, no passo 2, o item j de menor índice cuja demanda é maior do que zero. Se não existir tal j significa que a demanda de todos os itens foi atendida. Nesse caso a solução $\wp_1, \ldots, \wp_k$ é retornada e o procedimento pára. Caso contrário, buscamos determinar, no passo 3, o item i' de menor índice que possui demanda maior que zero e que ainda cabe em $\wp_k$.

Se existir tal item i', calculamos, no passo 3.1, $f = min(d_{i'}, \lfloor \frac{c(\wp_k)}{l_{i'}} \rfloor)$. Desviamos então o fluxo de execução para o passo 5 e buscamos uma solução fazendo chamadas recursivas ao procedimento PACKING, com f' variando de f até 0, tendo como parâmetros D_k, k, f' e i'. Essas chamadas recursivas podem levar a uma solução, caso em que o procedimento pára. Caso contrário, após cada chamada recursiva infrutífera atualizamos a demanda fazendo $d_{i'} = d_{i'} + f'$ e removemos de $\wp_k$ as f' ocorrências do item i'.

Se não existir tal item i', significa que nenhum item pode ser empacotado no espaço restante em $\wp_k$. Neste caso algumas condições devem ser satisfeitas antes que o padrão $\wp_k$ seja aceito. É preciso verificar se ainda é possível utilizar outra barra, ou seja, se $k < C$ e se o desperdício em $\wp_k$ é tal que, possivelmente, vai conduzir a uma solução que utilize mais do que C barras. Se a desigualdade

$$c(\wp_k) \leq \lfloor \frac{D_k}{C-k+1} \rfloor \tag{8}$$

não for satisfeita, rejeitamos o padrão. Pode-se mostrar que tal restrição não nos impede de encontrar uma solução, caso ela exista. Podemos impor mais uma restrição antes de aceitar um padrão. No procedimento PACKING, ao terminar de gerar o padrão $\wp_k$, o desperdício máximo do padrão $\wp_{k+1}$ é dado por $\lfloor \frac{D_k - c(\wp_k)}{C-k} \rfloor$. Dessa forma, resolvemos um problema da mochila onde os itens têm comprimento l_i, custo reduzido também l_i e demanda d_i $(i = 1, \ldots, m)$, de modo a descobrir se existe um padrão cujo desperdício seja no máximo $\lfloor \frac{D_k - c(\wp_k)}{C-k} \rfloor$. Caso não exista tal padrão, rejeitamos $\wp_k$.

No algoritmo FFDWB, cada padrão aceito é utilizado *uma vez*, portanto a frequência de cada item i contido no padrão não pode exceder d_i. Note ainda que o custo reduzido de cada item é um número inteiro. Essas duas peculiaridades ensejam modificações no algoritmo MOCHILA, proposto na Seção 3, modificações essas que limitam o espaço de busca, permitindo resolver o problema da mochila mais rapidamente. No passo 2 e no passo 3.2 do algoritmo MOCHILA, calculamos z_j fazendo $z_j = min(d_j, \lfloor (L - \sum_{i=1}^{j-1} l_i z_i)/l_j \rfloor)$. Com isto fazemos com que a frequência de cada item no padrão não exceda sua demanda. Além disso, como o custo reduzido y_i de cada item i é inteiro, podemos substituir a inequação $\sum_{j=1}^{i} y_j \bar{z}_j + \frac{y_{i+1}}{l_{i+1}}(L - \sum_{j=1}^{i} l_j \bar{z}_j) > M$, no passo 3, pela inequação mais forte $\sum_{j=1}^{i} y_j \bar{z}_j + \frac{y_{i+1}}{l_{i+1}}(L - \sum_{j=1}^{i} l_j \bar{z}_j) \geq M + 1$. Com tais modificações obtemos o algoritmo *MOCHILA especializado* (denotado por MOCHILAe).

Seja $t = L - \sum_{i=1}^{m} l_i z_i$ o comprimento desperdiçado na solução encontrada pelo algoritmo MOCHILAe. Qualquer padrão $\wp_k$ gerado no procedimento PACKING tem que possuir desperdício $c(\wp_k)$ tal que

$$t \leq \lfloor \frac{D_k - c(\wp_k)}{C - k} \rfloor. \tag{9}$$

No passo 3.2 executamos o algoritmo MOCHILAe com os parâmetros $L, l_1, \ldots, l_m, l_1, \ldots, l_m, d_1, \ldots, d_m$. No passo seguinte fazemos $t = L - \sum_{i=1}^{m} l_i z_i$. Podemos então verificar, no passo 3.4, se o padrão $\wp_k$ deve ser aceito ou rejeitado. Se $k < C$ e $c(\wp_k) \leq \lfloor \frac{D_k}{C-k+1} \rfloor$ e $t \leq \lfloor \frac{D_k - c(\wp_k)}{C-k} \rfloor$ o padrão $\wp_k$ deve ser aceito. Neste caso é feita uma chamada recursiva ao procedimento PACKING com os parâmetros $D_k - c(\wp_k), k+1, min(d_j, \lfloor \frac{L}{l_j} \rfloor)$ e j. Note que ao incrementarmos k, iniciamos a geração de um novo padrão, dessa forma o item j (que é o de menor índice com demanda maior que zero) deve ser empacotado $min(d_j, \lfloor \frac{L}{l_j} \rfloor)$ vezes neste novo padrão.

Caso a recursão contida no passo 3.4 não leve a uma solução, a chamada ao procedimento foi infrutífera. Devemos portanto executar um retrocesso. Para isso, no passo 4, fazemos $f = -1$ de forma que o laço contido no passo 5 não seja executado.

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 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. Uma vez compreendido o procedimento PACKING fica fácil entender o algoritmo FFDWB, que descrevemos a seguir.

Inicialmente colocamos os itens em ordem crescente usando como chave de ordenação, para cada item i, o número $min(d_i, \lfloor \frac{L}{l_i} \rfloor)$. Tal número representa o número máximo de vezes que o item i pode aparecer num padrão. Nos testes realizados constatamos ser este critério de ordenação usualmente mais vantajoso do que utilizar, por exemplo, ordem decrescente de comprimento, o que é feito no algoritmo FFDe.

Não está bem claro para nós por que esta heurística apresenta resultados melhores do que colocar os itens em ordem decrescente de comprimento, mas parece-nos que a explicação deste fenômeno está associada à idéia de limitar o número de ramificações nos primeiros níveis da árvore de busca. O algoritmo FFDWB consegue podar a árvore com maior frequência nos níveis de maior profundidade, portanto parece ser boa idéia construir a árvore de modo que os primeiros níveis possuam poucas ramificações e os níveis posteriores, que eventualmente não serão percorridos, possuam mais ramificações.

No passo 2 do algoritmo é calculado o desperdício D de uma solução que utiliza C barras fazendo-se $D = CL - \sum_{i=1}^{m} l_i d_i$, e é determinado f, que indica o número máximo de vezes que o item 1 pode aparecer no primeiro padrão. No passo seguinte o algoritmo busca uma solução fazendo chamadas ao procedimento PACKING com f' variando de f até 0, tendo como parâmetros $D, 1, f'$ e 1. Se uma solução for encontrada, o algoritmo pára no passo 2 do procedimento PACKING; caso contrário, no passo 4, o algoritmo retorna 0, significando que não existe solução que utilize no máximo C barras.

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 Coloque os itens em ordem crescente 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.

5 O Algoritmo HÍBRIDO

Estamos agora em condições de descrever o algoritmo híbrido que desenvolvemos, doravante referido como HÍBRIDO. No passo 1, inicializamos com o valor zero a variável v_{rl}, que representa o valor da solução ótima do problema relaxado, e a variável v_{ip} que representa o valor da solução inteira corrente. Aplicamos, no passo 2, o algoritmo SimplexGC para resolver a relaxação linear do problema. Na primeira iteração guardamos o valor da solução ótima x fazendo $v_{rl} = \sum_{i=1}^{m} x_i$. Nas demais iterações $\sum_{i=1}^{m} x_i$ nunca vai exceder v_{rl}, portanto v_{rl} não será mais modificado durante o algoritmo.

No passo 3 truncamos as variáveis fracionárias fazendo $x_i^* = \lfloor x_i \rfloor$ $(i = 1, \ldots, m)$, e acrescentamos o valor desta solução parcial inteira a v_{ip} fazendo $v_{ip} = v_{ip} + \sum_{i=1}^{m} x_i^*$. A seguir verificamos, no passo 4, se alguma parte da demanda foi atendida pelas variáveis x^*. Em caso afirmativo temos que $x_i^* > 0$ para algum $i = 1, ..., m$. Neste caso, retornamos, no passo 4.1, os padrões dados pelas colunas da matriz A com suas respectivas frequências, $x_1^*, \ldots, x_m^*$. Nos passos 4.2, 4.3 e 4.4, calculamos o problema residual. Primeiramente atualizamos a demanda fazendo $d_i = d_i - a_{ij} x_j^*$ $(i = 1, \ldots, m)$ e calculamos m', que representa a quantidade de itens cuja demanda ainda não foi totalmente atendida. Se $m' = 0$ então toda a demanda foi atendida portanto o algoritmo pára. Caso contrário, o problema residual deve ser resolvido. Para isso eliminamos os itens cuja demanda foi atendida, fazemos $m = m'$ e retornamos ao passo 2.

Se o teste executado no passo 4 falhar, ou seja, $x_i^* = 0$ $(i = 1, \ldots, m)$, então a solução obtida pelo método de geração de colunas, arredondada para baixo, não foi capaz de atender nenhuma parte da demanda. Neste caso, abandonamos o método de geração de colunas e utilizamos o algoritmo FFDWB para resolver este último problema residual.

No passo 5 fazemos $C = \lceil v_{rl} \rceil - v_{ip}$ e executamos o algoritmo FFDWB com os parâmetros $L, l_1, \ldots, l_m, d_1, \ldots, d_m, C$. Se o FFDWB não retornar 0, uma solução inteira utilizando C barras foi encontrada; neste caso retornamos, no passo 6, esta solução, dada por $\wp_1, \ldots, \wp_C$. Caso contrário, fazemos, no passo 7, uma nova chamada ao FFDWB com os parâmetros $L, l_1, \ldots, l_m$, $d_1, \ldots, d_m, C + 1$. Se o FFDWB não retornar 0, uma solução inteira utilizando $C + 1$ barras foi encontrada; neste caso retornamos, no passo 8, esta solução, dada por $\wp_1, \ldots, \wp_{C+1}$. Caso contrário, executamos o algoritmo FFDe com parâmetros $L, l_1, \ldots, l_m, d_1, \ldots, d_m$, retornamos a solução encontrada e paramos.

Algoritmo HÍBRIDO

Entrada: $(L, l_1, \ldots, l_m, d_1, \ldots, d_m)$.
Saída: Uma solução de: $\min \sum_{j=1}^{n} x_j$
sujeito a $Ax = d$
$x_j \geq 0$ e inteiro $\quad j = 1, \ldots, n$,
onde cada coluna j de A é tal que $\sum_{i=1}^{m} a_{ij} l_i \leq L$.

1 Faça $v_{rl} = 0$ e $v_{ip} = 0$.
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 $x_1^*, \ldots, x_m^*$.
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 Faça $m' = 0$.
4.4 Para $i = 1$ até m faça
4.3.1 Se $d_i > 0$ faça $m' = m' + 1$, $l_{m'} = l_i$ e $d_{m'} = d_i$.
4.5 Se $m' = 0$ então pare.
4.6 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 algoritmo FFDWB não retornou 0 então retorne $\wp_1, \ldots, \wp_C$ e pare.
7 Caso contrário, execute o algoritmo FFDWB com parâmetros $L, l_1, \ldots, l_m, d_1, \ldots, d_m, C+1$.
8 Se o algoritmo FFDWB não retornou 0 então retorne $\wp_1, \ldots, \wp_{C+1}$ e pare.
9 Caso contrário, execute o algoritmo FFDe com parâmetros $L, l_1, \ldots, l_m$, $d_1, \ldots, d_m$, retorne a solução encontrada e pare.

Note que executamos o algoritmo FFDWB para tentar achar uma solução que utilize apenas C ou $C + 1$ barras. Se o FFDWB não puder encontrar tais soluções, usamos o algoritmo FFDe para encontrar uma solução inteira para o problema residual. O motivo pelo qual não utilizamos o FFDWB para procurar soluções com $C + 2, C + 3, \ldots$ barras é explicado na próxima seção.

6 A Conjectura MIRUP

Seja I uma instância de um problema de programação linear inteira P, no qual queremos minimizar a função objetivo, $v(I)$ 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 [2] definiram a *Integer Round-Up Property (IRUP)* da seguinte forma:

Definição 6.1. *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 a IRUP é válida (ou se verifica) para uma instância I de um PLI se $v(I) = \lceil v_{rl}(I) \rceil$. Dyer, Frieze e McDiarmid [19] provaram que é $\mathcal{NP}$-difícil determinar se uma instância qualquer do PCE_1 possui ou não a IRUP.

Em 1985, Marcotte [18] mostrou que várias classes de instâncias do PCE_1 possuem a IRUP, No ano seguinte, no entanto, Marcotte [19] apresentou uma instância para a qual a IRUP não vale.

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

Para esta instância $v_{rl}(I) = 31,9666\ldots$ e $v(I) = 33$. Scheithauer e Terno [22], Gau [9], Schwerin e Wäscher [24] também exibiram instâncias com coeficientes pequenos, cuja diferença entre $v(I)$ e $v_{rl}(I)$ é maior que 1. Em nossos testes computacionais, também encontramos diversas instâncias cujo *gap* é maior que 1. Em [24] Schwerin e Wäscher apresentaram uma instância com 200 itens cujo *gap* entre o valor de uma solução inteira ótima e o valor de uma solução ótima da relaxação linear é 1,14435. Este é o maior *gap* conhecido até o momento. Estes resultados levaram Scheithauer e Terno [22] a definir a *Modified Integer Round-Up Property (MIRUP).*

Definição 6.2. *Um problema de programação linear inteira P (de minimização) possui a modified integer round-up property (MIRUP) se $v(I) \le \lceil v_{rl}(I) \rceil + 1$ para toda instância $I \in P$.*

Analogamente, dizemos que a MIRUP é válida (ou se verifica) para uma instância I de um PLI se $v(I) \le \lceil v_{rl}(I) \rceil + 1$. Em [22], Scheithuaer e Terno provaram que todos os problemas de corte unidimensional (i.e., seus correspondentes PLI) com $m \le 5$ possuem a MIRUP. Em experimentos computacionais realizados por diversos autores [23, 28], onde foram determinadas soluções inteiras ótimas, verificou-se que todas as instâncias testadas (incluindo as geradas aleatoriamente e aquelas mencionadas na literatura) apresentavam a MIRUP. Não se conhece até o momento nenhuma instância do PCE_1 que não possua a MIRUP. Em 1995, Scheithauer e Terno [23] apresentaram a seguinte conjectura.

Conjectura MIRUP: *O problema de corte unidimensional (1) possui a modified integer round-up property.*

Tal conjectura 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, o que é bastante improvável.

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 6.1. *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.*

Podemos dizer então que o algoritmo HÍBRIDO é um algoritmo *quase-exato*, se verdadeira a conjectura MIRUP. Dada uma instância I, chamemos de $H(I)$ o valor da solução encontrada pelo algoritmo HÍBRIDO e $OPT(I)$ o valor de uma solução inteira ótima. O Teorema 6.1 implica que o limite de desempenho assintótico do algoritmo HÍBRIDO é 1, se verdadeira conjectura MIRUP. Mais precisamente, vale o seguinte resultado.

Corolário 6.2. *Se a conjectura MIRUP for verdadeira então $H(I) \le OPT(I) + 1$ para toda instância I do PCE_1.*

7 Resultados Computacionais

Avaliamos o algoritmo híbrido descrito 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 na linguagem C e os testes executados numa estação Sun Sparc 1000, com dois processadores, 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.

7.1 Testes Aleatórios

Resolvemos 4000 instâncias aleatórias geradas utilizando o método delineado por Wäscher e Gau em [29]. Chamaremos tais instâncias de *instâncias de Wäscher e Gau*. Para gerar várias classes de instâncias aleatórias, variamos três parâmetros: o tamanho do problema, o intervalo de tamanho dos itens e o valor total da demanda.

O tamanho de uma instância do PCE_1 é expresso pelo valor m, que significa o número de itens. Nos testes realizados, m recebeu os valores 10, 20, 30, 40 e 50. Utilizamos $L = 10000$ unidades de comprimento. Os comprimentos dos itens foram modelados como variáveis aleatórias inteiras uniformemente distribuídas dentro do intervalo $[1, \frac{\beta L}{100}]$. O tamanho dos itens em relação ao tamanho L da barra afeta significativamente a dificuldade inerente ao problema e consequentemente a performance dos algoritmos propostos para o problema. Variamos o intervalo assumindo para β os valores 25, 50, 75 e 100.

Os valores d_i das demandas também foram tratadas como variáveis aleatórias inteiras. Elas foram geradas em duas etapas. Primeiramente calculamos a demanda total T fazendo $T = m\bar{d}$ ($\bar{d} \in I\!N$). Depois distribuímos a demanda total T pelos m itens. Podemos chamar $\bar{d}$ de *fator de multiplicação*. Variando este fator podemos mudar o caráter das instâncias, obtendo instâncias típicas do *bin-packing problem*, ao fixar $\bar{d}$ em valores pequenos, ou instâncias típicas do problema de corte de estoque, fixando $\bar{d}$ em valores grandes. Os valores que utilizamos para $\bar{d}$ foram 10 e 50.

Para distribuir a demanda total T pelos m itens, geramos os números aleatórios $R_1, \ldots, R_m$, uniformemente distribuídos dentro do intervalo [0,1], e então calculamos a demanda de cada item fazendo $d_i = max(1, \lfloor \frac{R_i}{R_1+\ldots+R_m} \rfloor T)$ para $i = 1, \ldots, m-1$ e $d_m = max(1, T - \sum_{i=1}^{m-1} d_i)$.

Combinando os diferentes valores destes três parâmetros definimos 40 classes de instâncias, classes estas caracterizadas pela tripla $(m, \beta, \bar{d})$. Para cada classe, 100 instâncias do problema foram geradas, perfazendo um total de 4000 instâncias. As instâncias foram geradas usando o algoritmo *CUTGEN1*, descrito em [11], e que implementa os procedimentos descritos nos três últimos parágrafos. Tal algoritmo recebe como entrada os parâmetros m, β, $\bar{d}$ e um valor usado para inicializar a semente da função que gera números pseudo-aleatórios. De forma a obter as mesmas instâncias utilizadas por Wäscher e Gau, inicializamos a semente usando o número obtido pela concatenação de m, β (com 3 dígitos) e $\bar{d}$. Por exemplo, ao gerar as instâncias da classe caracterizada por $m = 10$, $\beta = 75$ e $\bar{d} = 10$, o número utilizado para inicializar a semente foi 1007510.

Apesar de o algoritmo FFDWB ter capacidade de achar uma solução inteira ótima, se ela existir, os testes computacionais mostraram que em algumas instâncias o tempo de computação requerido pelo FFDWB era inaceitável. Por isso, em nossa implementação, limitamos em 250000 o número de nós que podem ser percorridos pelo FFDWB. Tal procedimento também foi adotado por Wäscher e Gau [29] ao restringir em 25000 o número de retrocessos efetuados pelo algoritmo MTP devido a Martello e Toth [20].

Na Tabela 1 apresentamos os resultados da aplicação do algoritmo HÍBRIDO às 4000 instâncias aleatórias geradas conforme descrevemos anteriormente. Nesta tabela, agrupamos as instâncias de classes que diferem apenas no parâmetro $\bar{d}$, para poder comparar com os resultados obtidos por Wäscher e Gau. Assim, cada linha da tabela representa os resultados obtidos na resolução de 200 instâncias.

A coluna *IRUP* representa o número de instâncias para as quais foi encontrada solução inteira que satisfaz a IRUP (portanto, solução ótima). A coluna *MIRUP* representa o número de instâncias para as quais foi encontrada solução inteira que satisfaz a MIRUP e não satisfaz a IRUP. A coluna *Colunas geradas* indica o número médio de colunas geradas pela aplicação do algoritmo SimplexGC em cada instância. A coluna *Tempo* informa o tempo médio requerido para resolver cada instância.

Comparamos a solução encontrada pelo algoritmo HÍBRIDO com a solução obtida pelo algoritmo FFD. Na coluna *Pior que FFD* indicamos o número de instâncias para as quais a solução encontrada pelo algoritmo HÍBRIDO foi pior que a solução encontrada pelo FFD. Informamos na coluna *Ganho sobre FFD* o ganho médio percentual da solução encontrada pelo algoritmo HÍBRIDO em relação à solução encontrada pelo FFD.

Finalmente, na coluna *Desperdício* indicamos o desperdício médio da solução encontrada pelo algoritmo HÍBRIDO. Consideramos como desperdício a parte não aproveitada das barras, independentemente da possibilidade de encontrar solução melhor, ou seja, o desperdício é dado pela razão entre o valor da solução inteira encontrada v_{ip} e $\frac{\sum_{i=1}^{m} d_i l_i}{L}$. Dessa forma, analisando a segunda linha da tabela, verificamos que ao resolver as 200 instâncias de tamanho m=10, cujos itens têm comprimento variando entre 1 e 7500 unidades de comprimento, desperdiçamos, em média, 15,084%, embora todas as soluções encontradas sejam ótimas. O propósito da coluna *Desperdício* é dar ao leitor uma idéia a respeito do desperdício inerente a essas classes de instâncias.

Pela Tabela 1 percebemos que a MIRUP foi verificada em todas as 4000 instâncias, sendo que em apenas 12 delas a IRUP não foi verificada. Isto significa que em no máximo 12 instâncias o algoritmo HÍBRIDO deixou de encontrar uma solução ótima. No entanto, a solução encontrada para estas 12 instâncias difere da solução ótima de no máximo 1, o que está de acordo com o Teorema 6.1.

Analisando a coluna *Tempo*, verificamos que o algoritmo HÍBRIDO encontra mais dificuldade para resolver instâncias onde a maior parte dos itens não apresenta comprimento muito pequeno nem muito grande em relação à barra. Estas instâncias estão caracterizadas por $\beta = 50$, ou seja, são as instâncias onde os itens têm no máximo a metade do comprimento da barra. Por outro lado, analisando a coluna *Ganho sobre o FFD*, percebemos que é justamente para $\beta = 50$ que o algoritmo HÍBRIDO apresenta maior vantagem em relação ao FFD. É interessante notar que, mesmo para a classe onde $m = 50$ e $\beta = 50$, o tempo médio necessário para resolver cada problema foi em torno de 20 segundos, o que, em termos práticos, é satisfatório.

Uma outra informação importante (embora não expressa na Tabela 1) é que a maior parte do tempo requerido para encontrar uma solução foi gasto na geração de colunas e não pelo algoritmo FFDWB. Em média, o tempo requerido pelo FFDWB foi inferior a 0,1 segundos.

Em todas as classes de instâncias o algoritmo HÍBRIDO mostrou ser superior ao FFD. Percebemos ainda que o desperdício diminui à medida que o tamanho do problema cresce, o que explica, em parte, por que o ganho sobre o FFD, de uma forma geral, também diminui quando o tamanho do problema aumenta.

A Tabela 2 apresenta uma comparação entre os resultados obtidos pelos algoritmos híbridos aqui propostos e diversos métodos encontrados na literatura, quando aplicadas às instâncias de Wäscher e Gau. Adotamos para estes métodos os nomes sugeridos por Wäscher e Gau [29] e indicamos as referências onde o leitor pode obter detalhes deste métodos. Dentre todos estes métodos, o algoritmo HÍBRIDO foi o que encontrou soluções ótimas para o maior número de instâncias.

Comparando com os resultados obtidos por Wäscher e Gau [29] ao utilizar diversas heurísticas para resolver as 4000 instâncias acima mencionadas, observamos que a qualidade das soluções encontradas pelo algoritmo HÍBRIDO é superior, visto que encontramos mais frequentemente uma

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

Tabela 1: Algoritmo HÍBRIDO aplicado às instâncias de Wäscher e Gau.

solução ótima.

Comparamos o ganho médio do algoritmo HÍBRIDO em relação ao FFD e o desperdício médio em função de m, respectivamente. Percebemos que o ganho médio em relação ao FFD e o desperdício médio tendem a diminuir e se estabilizar à medida que m cresce.

7.2 Testes Práticos

Apresentamos nesta seção 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 obras de construção 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 vergalhões de aço, originando 7 categorias de problemas. Obtivemos 209 instâncias, que chamaremos de *instâncias práticas*, agrupando a demanda de cada um destes 7 tipos de vergalhões em cada uma das 42 plantas de ferragem analisadas. A Tabela 4 mostra os resultados obtidos ao resolver estas instâncias usando o algoritmo HÍBRIDO.

Analisando a Tabela 3, percebemos que o algoritmo HÍBRIDO encontrou, dentro de um tempo bastante curto, soluções ótimas para todas as instâncias. Além disso, o algoritmo HÍBRIDO apresentou um ganho em relação ao FFD bastante satisfatório. O tempo médio gasto pelo algoritmo

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 [9]	797	792	790	779	763	3921
ROPT [29]	796	788	782	770	738	3874
CSTAOPT [26]	758	754	731	732	734	3709
RFFD [29]	779	758	743	726	695	3701
CSTAFFD [29]	752	746	715	716	701	3630
FFD	469	378	359	321	318	1845
BOPT [29]	428	321	262	251	210	1472
BRURED [21]	325	193	129	94	80	821
BRUSUC [26]	257	142	74	53	52	578
BRUSIM [29]	93	47	23	15	10	188

Tabela 2: Soluções inteiras ótimas obtidas por diversos algoritmos aplicados às instâncias de Wäscher e Gau.

Bitola	No de problemas	m médio	Tamanho dos itens	IRUP	Colunas geradas	Tempo (seg)	Ganho sobre FFD	Desper-dício
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.

FFDWB ao resolver estas 209 instâncias foi inferior a 0,01 segundos.

Por outro lado, nas categorias onde os itens têm tamanho médio ou grande (bitolas 16mm e 20mm), apesar de termos encontrado soluções ótimas para todas as instâncias, o desperdício médio foi bastante alto (acima de 10%). Este resultado é explicado, em parte, pelo fato de que as instâncias nestas categorias apresentam poucos itens (em média, $m \leq 13$).

Com o intuito de diminuir o desperdício, experimentamos agrupar todas as instâncias de cada categoria gerando apenas 7 instâncias, que chamaremos de *instâncias práticas agrupadas*. Com isto obtivemos instâncias maiores, esperando obter soluções com menor desperdício. Os resultados obtidos pelo algoritmo HÍBRIDO ao resolver estas 7 instâncias são apresentados na Tabela 4.

Soluções ótimas foram encontradas para todas as 7 instâncias. É interessante notar também que, mesmo para instâncias que podemos considerar grandes, o algoritmo HÍBRIDO apresentou tempo de execução aceitável. O tempo requerido pelo FFDWB para resolver cada uma destas 7 instâncias foi inferior a 0,01 segundos.

Na construção civil, o aço utilizado nas estruturas de concreto armado é adquirido por peso. Por isso, para ter uma idéia mais aproximada do desperdício, precisamos levar em conta o peso dos vergalhões de aço nas suas diferentes bitolas. A Tabela 5 apresenta a quantidade de aço especificada nas plantas e as quantidades utilizadas nas soluções apresentadas nas tabelas 3 e 4.

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 práticas agrupadas.

Vale salientar que, em obras deste tipo, esta construtora desperdiça, em média, algo em torno de 10% do aço. Este percentual é bom se comparado ao desperdício médio verificado na indústria de construção civil. Agrupando as instâncias, reduzimos o desperdício de aço a cerca de 1%, o que representa uma economia considerável.

Os resultados obtidos, tanto nos testes realizados com instâncias geradas aleatoriamente, quanto nos testes com instâncias práticas, demonstraram a eficiência do algoritmo HÍRIDO em termos de qualidade da solução e tempo de execução. Ademais, os resultados dos testes serviram para fortalecer a conjectura MIRUP.

			Tabela 3		Tabela 4	
Bitola	Peso (kg/m)	Quantidade 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.

8 Considerações Finais

O algoritmo HÍBRIDO que desenvolvemos reúne algoritmos de diversas naturezas, como o método Simplex com geração de colunas, o FFDWB e o FFDe. Deste fato deriva o seu nome. Esta abordagem diversificada tem se mostrado promissora, e tem sido bastante explorada em trabalhos recentes [26, 23, 29, 10]. 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. Nesse sentido, dizemos que o algoritmo HÍBRIDO é um algoritmo *quase-exato*.

Discorremos sobre resultados teóricos [22] e práticos [23, 28] que sugerem ser verdadeira a conjectura MIRUP. Os testes que realizamos com instâncias geradas aleatoriamente e instâncias práticas também servem para fortalecer esta conjectura. Assumindo que esta conjectura seja verdadeira, temos 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.

Repetimos os testes realizados por Wäscher e Gau [29] com 4000 instâncias (com até 50 itens) geradas aleatoriamente e constatamos que o algoritmo HÍBRIDO obteve desempenho superior ao de todos os algoritmos testados por estes autores. O algoritmo HÍBRIDO encontrou 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 que o algoritmo HÍBRIDO encontrou soluções ótima para todas estas instâncias.

O esquema utilizado no algoritmo HÍBRIDO proposto neste trabalho, pode ser adaptado para o problema de corte bi- e tridimensional. Para isto é necessário dispor de métodos para achar uma solução inicial, gerar novas colunas e resolver o problema residual final. Tais métodos são encontrados na literatura. Um desdobramento natural deste trabalho seria integrar estes métodos de forma a obter algoritmos capazes de encontrar soluções inteiras para o problema de corte bi- e tridimensional.

Referências

[1] B. S. Baker, *A new proof for the first-fit decreasing bin-packing algorithm*, Journal of Algorithms, 6 (1985), pp. 49–70.

[2] 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.

[3] 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.

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

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

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

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

[8] M. R. Garey, R. L. Graham, and D. S. Johnson, *On a number theoretic bin packing conjecture*, in Proc. 5th Hungarian Combinatorics Colloquium, Amsterdam, 1978, North-Holland, pp. 377–392.

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

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

[11] 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.

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

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

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

[15] ——, *Fast algorithms for bin packing*, J. Comput. Syst. Sci., 8 (1974), pp. 272–314.

[16] D. S. Johnson, A. Demars, J. D. Ullman, M. R. Garey, and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM Journal on Computing, 3 (1974), pp. 299–325.

[17] K. L. Krause, Y. Y. Shen, and H. D. Schwetman, *Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems*, J. Association Comput. Mach., 22 (1975), pp. 522–550.

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

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

[20] 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.

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

[22] 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.

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

[24] 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.

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

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

[27] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser, *Solving binary cutting stock problems by column generation and branch and bound*, Computational Optimization and Applications, 3 (1994), pp. 111–130.

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

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