UM ALGORITMO EXATO PARA O PROBLEMA DE EMPACOTAMENTO BIDIMENSIONAL EM FAIXAS

Carlos Eduardo de Andrade
Institudo de Computação - UNICAMP
Av. Albert Einstein, 1251, C.P.: 6176 CEP: 13084-971, Campinas/SP - Brasil
ce.andrade@gmail.com

Flávio Keidi Miyazawa
Institudo de Computação - UNICAMP
Av. Albert Einstein, 1251, C.P.: 6176 CEP: 13084-971, Campinas/SP - Brasil
fkm@ic.unicamp.br

Eduardo Cândido Xavier
Institudo de Computação - UNICAMP
Av. Albert Einstein, 1251, C.P.: 6176 CEP: 13084-971, Campinas/SP - Brasil
para@ic.unicamp.br

RESUMO

Neste artigo apresentamos um algoritmo branch-and-price para o Problema de Empacotamento Bidimensional em Faixas. O problema consiste em cortar uma faixa retangular em itens retangulares menores, utilizando a menor extensão da faixa possível. Utilizamos a regra de ramificação proposta por Vance (1994) e implementamos duas estratégias para geração de colunas. O algoritmo implementado foi testado em diversas instâncias conhecidas na literatura e apresentou resultados satisfatórios.

PALAVRAS CHAVE: Otimização; Corte; Empacotamento. Área: OC - Otimização Combinatória.

ABSTRACT

In this paper, we present a branch-and-price algorithm to Two-Dimensional Strip Packing Problem. This problem is to cut a rectangular strip in small rectangular items, minimizing the height of the strip. We use a branch rule proposed by Vance (1994) and implement two strategies to column generation. The proposed algorithm was tested with several well-known instances available in the literature and presented satisfactory results.

KEYWORDS: Optimization; Cut; Packing. Area: Combinatorial Optimization.

1. Introdução

Problemas de empacotamento, na sua forma geral, são aqueles que requerem que certos objetos, chamados de itens, sejam empacotados em outros, de tamanhos maiores, chamados de recipientes. Em algumas aplicações, em vez de empacotar, o objetivo é cortar. Note que os itens devem ser empacotados sem sobreposição. Problemas de corte e empacotamento aparecem freqüentemente na indústria e comércio como por exemplo indústrias de auto-peças, operações em portos e aeroportos, corte de um tecido visando a criação de peças de roupa, ou o corte de barras de aço para atender a diversas demandas de tamanhos e bitolas.

Um problema comum na indústria é o corte de um rolo de um determinado material para obtenção de vários itens menores, como peças de roupa ou metal. Devemos cortar o rolo para atender uma determinada demanda desses itens. Deste modo queremos obter estes itens utilizando a menor quantidade possível de material. Este problema é conhecido como Problema de Empacotamento Bidimensional em Faixas (do inglês *Two Dimensional Strip Packing*).

Problema (PEBF)**:** PROBLEMA DE EMPACOTAMENTO BIDIMENSIONAL EM FAIXAS
Seja S uma faixa de largura L e altura infinita e uma lista de itens retangulares $I = (r_1, \ldots, r_n)$, onde cada item $r_i = (l_i, a_i)$ é tal que $l_i \in (0, L]$, para $i = 1, \ldots, n$, l_i é a largura e a_i é a altura do item r_i. O objetivo é empacotar os itens de I em S com a menor altura possível.

Em geral, as máquinas utilizadas para o corte desses materiais operam apenas em sentido ortogonal (horizontal ou vertical) ao rolo, cortando de uma ponta a outra o material. Tais cortes são chamados de guilhotináveis. Os itens são obtidos com uma seqüência de estágios de cortes guilhotináveis que devem ser sucessivamente alternados no sentido horizontal e vertical. Uma restrição comum é uma limitação no número de estágios de cortes para se obter os itens finais. Tais restrições são comuns quando o custo envolvido nos cortes são relativamente altos. Neste artigo consideramos PEBF restrito a 2 estágios de corte, que denotamos por PEBF2. Consideramos também que o primeiro estágio de corte é sempre feito no sentido horizontal.

A Figura 1(a) mostra um padrão de corte guilhotinável e a Figura 1(b) um padrão guilhotinável com 2 estágios de corte. Note que um terceiro corte de ajuste pode ser necessário para eliminar a área inutilizada de um item. A Figura 1(c) mostra uma padrão que não pode ser guilhotinado.

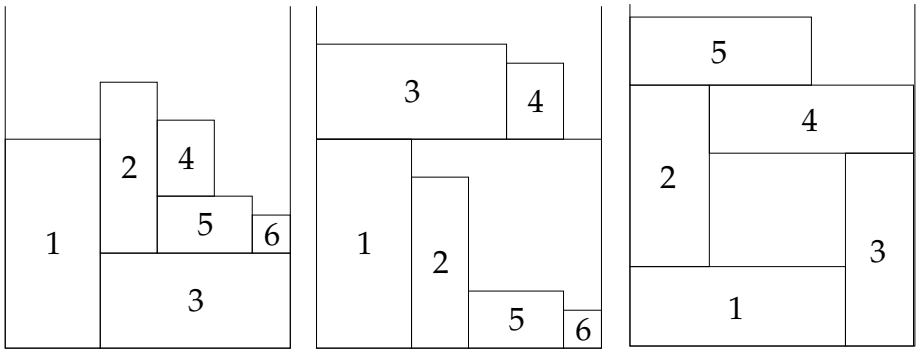

Figura 1: (a) Padrão Guilhotina; (b) Padrão com 2 estágios; (c) padrão não-guilhotinável

O PEBF2 é um problema NP-difícil, já que podemos reduzir o Problema do Empacotamento Unidimensional a ele (Garey e Johnson (1979)). Desta maneira, a não ser que $\mathrm{P} = \mathrm{NP}$, o PEBF2 não pode ser resolvido na otimalidade em tempo polinomial.

Várias abordagens são utilizadas para a resolução do PEBF. Hopper e Turton (2001a) apresentam uma revisão de várias heurísticas utilizadas em problemas de empacotamento em faixa, enfatizando algoritmos genéticos. Lodi, Martello e Vigo (2002) apresentam um *survey* contendo

vários algoritmos aproximados, entre eles o *Next-fit Decresing Height* (NFDH) e *First-fit Decresing Height* (FFDH), e o *Best-fit Decresing Height* (BFDH). Um esquema de aproximação para o PEBF foi apresentado por Kenyon e Rémila (2000).

Neste artigo, apresentamos um algoritmo *branch-and-price* para o PEBF2. Pelo conhecimento dos autores, este é o primeiro artigo que apresenta um algoritmo de *branch-and-price* para o PEBF2. A principal motivação em usar este método está na qualidade dos limitantes inferiores obtidos pela relaxação linear associada que muitas vezes são próximas da solução ótima. Desta forma esperamos que a árvore de busca não seja tão grande.

A Seção 2 mostra duas formulações do PEBF2, uma onde o *gap* de integralidade é grande e outra para a geração de colunas. A Seção 3 mostra as considerações quanto as estratégias de ramificação e seus impactos na geração de colunas e cálculo dos limitantes superiores. A Seção 4 traz alguns detalhes na implementação deste algoritmo e a Seção 5 os resultados dos experimentos computacionais.

2. Formulações

A primeira formulação para problemas de corte e empacotamento é devida a Kantorovic (1960). Posteriormente, muitos trabalhos apresentaram diferentes formulações às mais diversas variantes destes problemas.

Abaixo apresentamos a formulação proposta por Lodi, Martello e Vigo (2004). Esta formulação contém 2 conjuntos de variáveis. Sem perda de generalidade, assumimos que $a_1 \geq a_2 \geq \ldots \geq a_n$. Assumimos $n = |I|$ possíveis níveis, cada um associado a um item r_i que inicializa este, tendo, portanto, altura correspondente a_i. O primeiro conjunto de variáveis indica se um item inicia ou não um nível:

$$y_i = \begin{cases} 1 & \text{se } r_i \text{ inicializa o nível } i, \\ 0 & \text{caso contrário} \end{cases} \qquad \text{para } i = 1, \ldots, n \tag{1}$$

e o segundo indica se um item j está empacotado em um nível i:

$$x_{ij} = \begin{cases} 1 & \text{se } r_j \text{ está empacotado no nível } i, \\ 0 & \text{caso contrário} \end{cases} \qquad \text{para } i = 1, \ldots, n \text{ e } j > i\,. \tag{2}$$

A formulação é apresentada a seguir:

$$\text{minimize} \quad \sum_{i=1}^{n} a_i y_i \tag{3}$$

$$\text{sob as restrições} \quad \sum_{i=1}^{j-1} x_{ij} + y_i = 1 \qquad \text{para } j = 1, \ldots, n \tag{4}$$

$$\sum_{j=i+1}^{n} l_j x_{ij} \leq (L - l_i) y_i \qquad \text{para } i = 1, \ldots, n-1 \tag{5}$$

$$y_i \in \{0, 1\} \qquad \text{para } i = 1, \ldots, n \tag{6}$$

$$x_{ij} \in \{0, 1\} \qquad \text{para } i = 1, \ldots, n-1 \text{ e } j > i\,. \tag{7}$$

A Função Objetivo (3) minimiza a altura do empacotamento, a Restrição (4) força que um item seja empacotado apenas uma vez, ou iniciando um nível ou como parte deste nível, e a Restrição (5) limita a ocupação de itens em um nível por sua largura. As Restrições (6) e (7) garantem a integralidade das variáveis.

Esta formulação esbarra em uma dificuldade: o *gap* de integralidade é igual a $1/2$ (Lodi, Martello e Vigo (2002)). Uma maneira de "apertar" a formulação é aplicar a decomposição de Dantzig-Wolfe. Gilmore e Gomory (1961) foram pioneiros na utilização desta técnica aplicando-a ao problema de empacotamento unidimensional.

Reformulando o PEBF2 temos o seguinte problema linear, o qual chamamos de mestre:

$$\begin{aligned} \text{minimize} \quad & \sum_{P\in\mathcal{P}} a_P\lambda_P & & \\ \text{sob as restrições} \quad & \sum_{P\in\mathcal{P}} P_i\lambda_P = 1 & \forall i \in \{1,\ldots,n\} & \qquad (8)\\ & \lambda_P \in \{0,1\} & \forall P \in \mathcal{P} & \end{aligned}$$

onde $P = (P_1, \ldots, P_n)$ é um vetor binário representando a configuração de um nível. O conjunto $\mathcal{P}$ contém todos os níveis P que satisfazem $\sum_{j=1}^{n} P_j l_j \leq L$ e definimos $a_P = \max(P_j \cdot a_j)$ com $j \in \{1,\ldots,n\}$. A variável λ_P indica se o nível é utilizado ou não na solução e P_i se o item i está empacotado no nível P. Esta formulação é mais restritiva uma vez que as soluções fracionárias que não são combinações convexas de soluções inteiras de problemas da mochila, não são viáveis segundo Barnhart *et al.* (1998). Embora mais restritiva, esbarramos em uma formulação de tamanho exponencial, devido ao imenso número de níveis possíveis. Mas mesmo assim, podemos resolver de forma razoavelmente rápida a relaxação desta formulação utilizando a técnica de geração de colunas. Para uma base do Problema (8), temos um vetor dual associado que denotamos por π. Na geração de colunas temos que resolver o seguinte subproblema da mochila:

$$\begin{aligned} M_k = \quad \text{maximize} \quad & \sum_{i\in I^k} \pi_i P_i & & \\ \text{sob as restrições} \quad & \sum_{i\in I^k} l_i P_i \leq L & & \qquad (9)\\ & P_i \in \{0,1\} & \forall i \in I^k & \end{aligned}$$

onde o conjunto $I^k = \{i \in I : a_i \leq A_k\}$, P_i é uma variável que indica se o item i está empacotado no padrão e π_i é um valor dual associado ao item i. Para cada possível altura A_k de um nível, devemos resolver o subproblema M_k correspondente. A coluna de melhor custo reduzido deve ser devolvida para o problema mestre. Note que número de altura possíveis é limitado pelo número de itens.

Note que, embora seja NP-difícil, o problema da mochila associado a geração de colunas admite um algoritmo de programação dinâmica pseudo-polinomial de tempo $\mathrm{O}(nL)$ onde n é o número de itens e L o tamanho da mochila (originalmente proposto por Gilmore e Gomory (1967) e exposto por Martello e Toth (1990)).

3. Algoritmo de *Branch-and-Price* para o PEBF2

Utilizando a Formulação (8) podemos construir um algoritmo de *Branch-and-Price* para obtenção de soluções exatas.

O algoritmo primeiramente resolve a relaxação do Programa Linear (8) utilizando a técnica de geração de colunas. Se a solução tiver variáveis fracionárias, devemos inserir restrições sobre estas variáveis de modo que a solução fracionária corrente seja descartada. Este processo, conhecido como ramificação, cria subproblemas que devem ser reotimizados utilizando a técnica de geração de colunas.

Um dos grandes desafios em algoritmos de *Branch-and-Price* é a escolha de estratégias de ramificação que garantam que o programa linear mestre consiga ser resolvido pela técnica de geração de colunas. Na subseção a seguir, discutiremos em maior detalhes como a ramificação é feita em nosso algoritmo.

3.1. Ramificação

A forma mais simples de ramificação, já que as variáveis são binárias, é ajustar os limites destas para zero em um ramo, digamos o esquerdo, e para 1 no outro ramo, digamos direito. Seja λ_P a variável onde ajustamos os limites. Então, $\lambda_P = 0$ no ramo esquerdo indicando que o nível P não pode ser usado neste ramo, e $\lambda_P = 1$ no ramo direito o que força a utilização de P. Entretanto, segundo Barnhart *et al.* (1998), é possível (e muito provável) que o nível P seja gerado novamente na próxima geração de colunas devido ao seu custo reduzido ter se tornado atrativo. Desta maneira, supondo que estamos no j-ésimo nível da árvore, poderemos estar em uma situação onde precisamos gerar a j-ésima melhor coluna, o que pode demandar muito tempo. Outra dificuldade neste esquema de ramificação é que o ajuste dos limites das variáveis do problema mestre requer modificações no subproblema de geração de colunas que geralmente destroem a estrutura que é explorada ou necessária para resolução eficiente destes subproblemas, como verificado por Vanderbeck (2000).

Para contornar estas dificuldades, Vance (1994) propõe uma regra de ramificação baseada no trabalho de Ryan e Foster (1981) e Barnhart *et al.* (1998). Esta regra se basea na seguinte proposição cuja demonstração pode ser encontrada em Barnhart *et al.* (1998):

Proposição 3.1. *Seja X uma matriz binária e λ uma solução básica tal que $X\lambda = 1$. Se pelo menos um dos componentes de λ é fracionário, então existem duas linhas r e s do problema mestre tal que*

$$0 < \sum_{k:x_{rk}=1,x_{sk}=1} \lambda_k < 1\,,$$

onde $x_{ik} = 1$ indica que o item i é coberto pela coluna k.

Baseadas nesta proposição, temos as seguintes restrições:

$$\sum_{k:x_{rk}=1,x_{sk}=1} \lambda_k = 1 \tag{10}$$

e

$$\sum_{k:x_{rk}=1,x_{sk}=1} \lambda_k = 0\,. \tag{11}$$

A Restrição (10) faz com que as linhas r e s sejam cobertas por uma mesma coluna, enquanto que a Restrição (11) faz com que as linhas sejam cobertas por colunas diferentes.

Na resolução do problema com a Restrição (10) são permitidos apenas padrões cujos itens r e s ou estejam juntos no padrão ou nenhum apareça. Na resolução com a Restrição (11) são geradas colunas onde apenas um dos itens aparece no padrão ou nenhum deles.

3.2. Geração de Colunas

A geração de colunas deve ser feita para todas alturas possíveis de níveis. No nó inicial da árvore é utilizado o algoritmo de programação dinâmica da mochila diretamente sobre a Formulação (9). Mas nos nós subjacentes, devido as restrições impostas na ramificação pelas Equações (10) e (11), a geração de colunas não pode ser feita por esse algoritmo sem alterações.

As restrições impostas pela Equação (10) podem ser facilmente implementadas criando-se um novo item cuja largura é a soma dos itens considerados na ramificação e a altura é a maior entre os dois itens. Estes são eliminados em *prol* do novo item. Seja r e s os itens considerados, a

formulação do subproblema para cada altura A_k de nível é a seguinte:

$$\begin{aligned} M_k = \quad & \text{maximize} && \sum_{i \in S} \pi_i P_i \\ & \text{sob as restrições} && \sum_{i \in S} l_i P_i \leq L \\ & && P_i \in \{0,1\} \qquad\qquad \forall i \in S \end{aligned} \tag{12}$$

onde

$$S = \{i \in (I \setminus \{r,s\}) \cup \{m\} : a_i \leq A_k\}$$

e m é o novo item tal que:

$$\begin{aligned} l_m &= l_r + l_s \\ a_m &= \max(a_r, a_s) \\ \pi_m &= \pi_r + \pi_s \, . \end{aligned}$$

Como substituímos dois itens por um de mesma largura e mesmo valor, o algoritmo da mochila não precisa ser alterado.

As restrições impostas pela Equação (11), que dizem que os itens devem aparacer separados, são mais complicadas. Elas remetem a seguinte formulação para cada altura A_k:

$$\begin{aligned} M_k = \quad & \text{maximize} && \sum_{i \in I^k} \pi_i P_i \\ & \text{sob as restrições} && \sum_{i \in I^k} l_i P_i \leq L \\ & && P_r + P_s \leq 1 \qquad\qquad r,s \in I^k \\ & && P_i \in \{0,1\} \qquad\qquad \forall i \in I^k \end{aligned} \tag{13}$$

onde $I^k = \{i \in I : a_i \leq A_k\}$ e, r e s são os itens considerados na ramificação.

A Restrição (13), que chamamos de restrição de aresta, indica que os itens r e s não devem estar no mesmo padrão.

Note que após várias ramificações, podemos ter várias restrições de aresta impostas a um nó na árvore de ramificações. Denotamos por B o conjunto de pares de itens selecionados em ramificações dadas pela Equação (11).

O subproblema que possui restrições de aresta é resolvido utilizando-se um resolvedor de programação inteira, ou através da resolução de vários problemas da mochila, um para cada uma das possíveis combinações de itens que respeitem as restrições de aresta (13). Ambas abordagens são lentas e consomem o maior tempo do algoritmo. O teste destas combinações é custoso: embora um conjunto de itens que respeitem as restrições de aresta (13) possa ser utilizado pelo algoritmo da mochila para gerar uma coluna, devemos testar todas combinações válidas possíveis. Seja o grafo G onde os vértices representam os itens em B, e as arestas são dadas pelo par de itens $(r,s) \in B$ dados por uma restrição (13). Encontrar conjuntos de itens válidos equivale a encontrar todos conjuntos independentes em G. Potencialmente, o número de conjuntos pode atingir $2^{|B|}$. Testar todas combinações pode tornar-se proibitivo mesmo para um conjunto B de tamanho moderado.

3.3. Limitantes Superiores

Como o PEBF2 é um problema de minimização, seus limitantes superiores representam soluções válidas calculadas através de heurísticas ou algoritmos aproximados. Consideramos aqui quatro algoritmos dos quais um é utilizado para o cálculo do limitante superior inicial apenas e os outros três para o cálculo de limitantes superiores para cada um dos nós da árvore.

O algoritmo utilizado para cálculo do limitante superior inicial é conhecido como Mochila Gulosa e foi proposto por Lodi, Martello e Vigo (1999). Este é um algoritmo guloso onde a cada iteração, o item mais alto é empacotado em um nível, e o restante deste é completado por um algoritmo que resolve o problema da mochila de forma exata. A razão da utilização deste algoritmo apenas no nó inicial da árvore é que o mesmo não pode tratar eficientemente as restrições inseridas ao longo das ramificações da árvore, em especial restrições do tipo (13).

Os algoritmos utilizados para o cálculo de limitantes superiores nos demais nós da árvore são clássicos da literatura: *Next-fit Decresing Height* (NFDH), *First-fit Decresing Height* (FFDH) e *Best-fit Decresing Height* (BFDH) (podemos encontrar referências em Lodi, Martello e Vigo (2002)). Ambos algoritmos podem ser implementados em tempo $\mathrm{O}(n \log n)$ em sua forma básica. O NFDH (respectivamente FFDH e BFDH) gera um empacotamento cuja altura é no máximo 2 (respectivamente 1,7 e 1,7) vezes a altura ótima, mais um termo aditivo proporcional a altura do maior item.

Para calcular os limitantes de nós internos da árvore, o NFDH, o FFDH e o BFDH devem ser modificados para atenderem as restrições de aresta. Esta modificação consiste simplesmente em verificar a compatibilidade do item a ser empacotado com os itens do nível tratado no momento. Esta modificação torna o algoritmo $\mathrm{O}(n^2)$ e não mais garante os fatores de aproximação.

4. Implementação

4.1. Ramificação

A regra de ramificação implementada segue da proposição da Seção 3.1. O algoritmo procura por dois itens com os quais seja possível realizar a ramificação. Caso estes itens não possam ser encontrados, o nó atual corresponde a uma solução inteira. Se encontrados, são criados dois nós filhos tal que, em um nó, designado como filho esquerdo, o limite superior é ajustado para zero em todas as colunas que contém um item e não contém o outro, e no outro nó, designado como filho direito, o limite superior é ajustado para zero nas colunas que contém ambos os itens. Esta informação é repassada para os geradores de colunas, garantindo que colunas geradas em cada ramo não inflijam as regras impostas até então. Note que o domínio dessas variáveis é $\{0, 1\}$ o que implica na proibição das variáveis com limites ajustados para zero.

Utilizamos duas abordagens de escolhas dos itens. Em uma delas, dois itens quaisquer que satisfaçam a Proposição 3.1, podem ser escolhidos. Essa abordagem pode nos levar a escolha de itens já considerados em ramificações anteriores o que pode ser uma vantagem para geração de colunas no filho direito quando usamos o teste das combinações, já que torna mais denso o grafo resultante das restrições de aresta e diminui o número de conjuntos independentes. Quando utilizamos o resolvedor, isso pode ser uma desvantagem, pois um grafo denso pode ser um complicante para as heurísticas deste. Em vista disso, a segunda abordagem escolhe itens que ainda não foram considerados em ramificações anteriores. Assim, promovemos a formação de restrições "independentes" quanto aos itens que as compõe, formando uma estrutura similar a blocos, cujo resolvedor pode tirar proveito. Em contrapartida, temos um grande número de conjuntos independentes a serem testados.

Durante a busca na árvore, os nós que não contêm restrições de aresta (13) são resolvidos primeiro, já que sua resolução pelo algoritmo da mochila é rápida.

4.2. Geração de Colunas

A geração de colunas deve ser feita para cada uma das possíveis alturas de níveis, e foi divida em duas partes. A primeira trata-se da geração em nós onde não existem restrições de aresta. Nestes nós, o algoritmo de programação dinâmica do problema da mochila é utilizado para cada possível altura. A coluna gerada com melhor custo reduzido é devolvida.

A geração de colunas é lenta nos nós que contém restrições de aresta. Utilizamos duas abordagens: em uma, é construído um problema linear inteiro que é passado para o resolvedor. A segunda abordagem, são geradas todas combinações de itens que satisfazem as restrições de arestas. Para cada uma destas combinações, geramos uma coluna utilizando o algoritmo de programação dinâmica para o problema da mochila. Se x é o número de itens no conjunto B de pares de itens que formam as restrições de aresta, então o número máximo de combinações de itens quer satisfazem as restrições de aresta é 2^x. Utilizamos um parâmetro que permite escolher o número de conjuntos a serem testados. Se o número de conjuntos for maior que este, e não conseguimos gerar uma coluna com custo reduzido negativo, utilizamos o resolvedor para gerar a coluna. Na nossa implementação, assim que geramos um coluna com custo reduzido negativo, a devolvemos. Assim, nem todas alturas são testadas.

4.3. Limitantes Superiores

Para obtenção de limitantes superiores utilizamos os algoritmos da mochila gulosa, NFDH, FFDH e BFDH.

A mochila gulosa só é utilizada para cálculo do primeiro limitante superior no nó raiz da árvore e utiliza a implementação da programação dinâmica da mochila. O mochila gulosa apresentou resultados muito parecidos com o FFDH e BFDH. Desta maneira, modificamos o cálculo dos valores dos itens, não apenas considerando sua área mas também uma pequena depreciação nestes baseada na largura do item em relação a capacidade da mochila. Para cada item, calculamos a porcentagem de sua ocupação na mochila e as que ficaram abaixo de um limiar definido, foi aplicado uma função de amortização do valor (área) do item. Seja $0 < B < 1$ o limiar e $0 < PT_i \leq 1$ a porcentagem de ocupação do item i em relação ao tamanho da mochila. Foram aplicadas a função linear $v(i) = area(i) \cdot (PT_i/B)$, a função quadrática $v(i) = area(i) \cdot (PT_i/B)^2$ e a função exponencial $v(i) = area(i) \cdot \epsilon^{10 \cdot (PT_i - B)}$.

As implementações do NFDH, FFDH e BFDH são similares e levam a um algoritmo $\mathrm{O}(n^2)$ como mostrado na Seção 3.3 por causa das restrições de aresta. Na prática, podemos assumir uma complexidade menor, já que o tempo de execução está diretamente ligado ao número de restrições de arestas.

5. Experimentos Computacionais

A implementação foi feita em C++ utilizando o *framework* BCP da infraestrutura COIN. O BCP é um *framework* para *branch-and-cut-and-price* que visa facilitar a implementação de algoritmos deste tipo provendo gerenciamento automático da árvore de ramificação, cortes e colunas. O resolvedor utilizado foi o Xpress-MP®, versão 16.10.03. A máquina utilizada foi um Pentium® 4 2GHz com 1GB de memória RAM.

As instâncias de teste foram adaptadas da ORLIB que contêm apenas algumas instâncias para o PEBF. Estas instâncias nomeadas de `ht` são originárias do trabalho de Hopper e Turton (2001b). Utilizamos outras instâncias da ORLIB propostas para o Problema de Empacotamento Bidimensional e o Problema de Corte Bidimensional. Neste caso a largura corresponde a largura dos recepientes destas instâncias. Utilizamos as instâncias descritas por Christofides e Whitlock (1977), chamadas de `gccut` e instâncias descritas por Beasley (1985) chamadas de `gcut`. As instâncias descritas por Bengtsson (1982) chamadas de `beng` foram sugeridas pelo trabalho de Martello, Monaci e Vigo (2003) e não constam na ORLIB. O número de instâncias de teste é de 27 para o algoritmo em si, e 46 instâncias para os algoritmos de cálculo dos limitantes superiores.

Primeiramente avaliamos os algoritmos para cálculo dos limitantes superiores. O NFDH, FFDH e BFDH apresentaram os comportamentos já esperados (FFDH e BFDH melhores que o NFDH, com ligeiro ganho no BFDH). O algoritmo da mochila gulosa apresentou ganho em 14 das 46 instâncias (30,34%), nenhuma melhoria em 18 delas (39.13%) e perdeu em 14 (30,34%). Foram implementados também a depreciação baseada na largura do item em relação a capacidade da mochila. Utilizamos a relação de 5%, 10%, 15%, 20%, 25% e 30%, aplicados as funções descritas na Seção 4.3. Em apenas um instância (`beng04`) obtivemos melhoria. Portanto, optamos em não aplicar a função de amortização.

Dados esses resultados, optamos por aplicar o algoritmo da mochila gulosa para gerar o limitante superior inicial e o algoritmo BFDH para gerar os limitantes nos demais nós. Na Tabela 1 apresentamos os resultados do algoritmo. A coluna (`LI`) apresenta os limitantes inferiores correspondentes ao teto do valor ótimo da relaxação do programa linear mestre. A coluna (`LS`) apresenta o valor do limitante superior calculado pelo algoritmo da mochila gulosa. A coluna (`Obj`) apresenta os valores encontrados, em no máximo 1 hora de execução. A coluna (`Res.`) corresponde a execução do algoritmo utilizando o resolvedor de programação linear inteira para gerar colunas em nós com restrições de aresta, e a coluna (`Comb.`) apresenta os resultados utilizando a estratégia de combinações. Esta estratégia considera no máximo 32 combinações. Caso nenhuma destas retorne uma coluna de custo reduzido negativo, o resolvedor é utilizado. A coluna (`Gap`) indica o *gap* entre o valor da solução e o limitante inferior calculado através da equação $100 \cdot (Obj - LI)/Obj$. A coluna (`Opt?`) indica se foi provada a otimalidade em uma 1 hora de execução. As instâncias marcadas com asterisco correspondem àquelas que foram resolvidas na otimalidade.

Tabela 1: Resultados do algoritmo (tempo em segundos)

				Obj		*Gap*		Opt?		Total de nós		Tempo Total (s)	
Nome	Itens	LI	LS	Res.	Comb.	Res.	Comb.	Res.	Comb.	Res.	Comb.	Res.	Comb.
beng01	20	33	36.00	36.00	36.00	9.09	9.09	*	*	4167	3977	302.18	226.62*
beng02	40	60	62.00	61.00	61.00	1.67	1.67	*	*	797	1525	244.79	249.15
beng04	80	108	113.00	113.00	113.00	4.63	4.63			7599	11637	3600.04	3600.77
beng05	40	37	41.00	40.00	40.00	8.11	8.11			13307	18713	3600.18	3600.02
cgcut1	7	12	14.00	14.00	14.00	16.67	16.67	*	*	5	5	0.04	0.03
cgcut2	10	46	53.00	51.00	51.00	10.87	10.87	*	*	29	29	1.71	0.36*
gcut01	10	1016	1016.00	1016.00	1016.00	0.00	0.00	*	*	1	1	0.01	0.01
gcut02	20	1262	1347.00	1262.00	1262.00	0.00	0.00	*	*	9	9	0.22	0.19
gcut03	30	1810	1899.00	1810.00	1810.00	0.00	0.00	*	*	1	1	0.18	0.18
gcut05	10	1360	1375.00	1360.00	1360.00	0.00	0.00	*	*	1	1	0.04	0.04
gcut06	20	2792	2862.00	2862.00	2862.00	2.51	2.51	*	*	209	199	13.97	17.81
gcut07	30	4894	4979.00	4979.00	4979.00	1.74	1.74	*	*	1247	1073	185.05	221.38
gcut08	50	6149	6301.00	6168.00	6168.00	0.31	0.31	*	*	3	3	2.03	2.15
gcut09	10	2509	2664.00	2646.00	2646.00	5.46	5.46	*	*	31	31	0.62	0.39
gcut10	20	6167	6539.00	6167.00	6167.00	0.00	0.00	*	*	1	1	0.15	0.16
gcut11	30	7298	7571.00	7298.00	7298.00	0.00	0.00	*	*	1	1	0.60	0.64
gcut12	50	14943	15266.00	14943.00	14943.00	0.00	0.00	*	*	1	1	1.34	1.40
gcut13	32	5090	5500.00	5398.00	5398.00	6.05	6.05			2919	1587	3601.07	3606.87
ht_c1_01	16	26	28.00	27.00	27.00	3.85	3.85	*	*	3	3	0.25	0.10
ht_c1_02	17	29	29.00	29.00	29.00	0.00	0.00	*	*	1	1	0.11	0.12
ht_c1_03	16	23	23.00	23.00	23.00	0.00	0.00	*	*	1	1	0.16	0.17
ht_c2_01	25	20	22.00	20.00	20.00	0.00	0.00	*	*	1	1	1.26	1.26
ht_c2_02	25	33	34.00	34.00	34.00	3.03	3.03	*	*	3	3	1.86	1.70
ht_c2_03	25	23	23.00	23.00	23.00	0.00	0.00	*	*	1	1	1.42	1.41
ht_c3_01	28	37	41.00	40.00	40.00	8.11	8.11	*	*	85	71	145.77	15.22*
ht_c3_02	29	41	42.00	42.00	42.00	2.44	2.44	*	*	3	3	6.58	3.12*
ht_c3_03	28	42	43.00	43.00	43.00	2.38	2.38	*	*	5	5	7.21	2.64*

A média do *gap* da solução obtida pelo algoritmo e o limitante inferior foi de 3,21%, mostrando a qualidade do limitante inferior. Em 5 das 27 instâncias testadas (marcadas com * na coluna `Tempo Total`), o algoritmo utilizando a técnica combinatória, obteve um ganho expressivo em relação ao algoritmo que utiliza apenas o resolvedor para gerar colunas com restrições de aresta. Nas instâncias em que o algoritmo combinatório foi pior, a diferença de tempo em relação ao algoritmo com resolvedor foi pequena.

A Tabela 2 apresenta detalhes sobre o número de colunas gerados pelo algoritmo. A coluna (`Vars Res.`) apresenta o número de colunas geradas utilizando o resolvedor e a coluna (`Vars Comb.`) o número de colunas geradas utilizando a estratégia de combinações. Em cada uma dessas colunas, (`KN`) indica o número de colunas geradas pelo algoritmo de programação dinâmica em nós sem restrições de aresta, (`MIP`) e o número de colunas utilizando o resolvedor e (`Comb`) o número

de colunas utilizando a estratégia de combinações.

Tabela 2: Número de colunas (tempo em segundos)

		Opt?		Cols Res.			Cols Comb.			Total Cols		Tempo Total	
Nome	Itens	Res.	Comb.	KN	MIP	Comb.	KN	MIP	Comb.	Res.	Comb.	Res.	Comb.
beng01	20	*	*	63	15952	0	63	6369	7713	16015	14145	302.18	226.62
beng02	40	*	*	137	5498	0	137	3140	4403	5635	7680	244.79	249.15
beng04	80			425	33278	0	425	23660	24814	33703	48899	3600.04	3600.77
beng05	40			260	84713	0	260	51060	71587	84973	122907	3600.18	3600.02
cgcut1	7	*	*	10	0	0	10	0	0	10	10	0.04	0.03
cgcut2	10	*	*	38	106	0	38	0	94	144	132	1.71	0.36
gcut01	10	*	*	0	0	0	0	0	0	0	0	0.01	0.01
gcut02	20	*	*	38	6	0	38	0	6	44	44	0.22	0.19
gcut03	30	*	*	37	0	0	37	0	0	37	37	0.18	0.18
gcut05	10	*	*	14	0	0	14	0	0	14	14	0.04	0.04
gcut06	20	*	*	31	273	0	31	88	214	304	333	13.97	17.81
gcut07	30	*	*	30	1523	0	30	610	705	1553	1345	185.05	221.38
gcut08	50	*	*	98	6	0	98	0	9	104	107	2.03	2.15
gcut09	10	*	*	12	19	0	12	0	17	31	29	0.62	0.39
gcut10	20	*	*	22	0	0	22	0	0	22	22	0.15	0.16
gcut11	30	*	*	43	0	0	43	0	0	43	43	0.60	0.64
gcut12	50	*	*	40	0	0	40	0	0	40	40	1.34	1.40
gcut13	32			294	29166	0	294	5251	6769	29460	12314	3601.07	3606.87
ht_c1_01	16	*	*	30	20	0	30	0	15	50	45	0.25	0.10
ht_c1_02	17	*	*	51	0	0	51	0	0	51	51	0.11	0.12
ht_c1_03	16	*	*	76	0	0	76	0	0	76	76	0.16	0.17
ht_c2_01	25	*	*	203	0	0	203	0	0	203	203	1.26	1.26
ht_c2_02	25	*	*	242	14	0	242	0	3	256	245	1.86	1.70
ht_c2_03	25	*	*	220	0	0	220	0	0	220	220	1.42	1.41
ht_c3_01	28	*	*	250	4088	0	250	0	1581	4338	1831	145.77	15.22
ht_c3_02	29	*	*	330	94	0	330	0	30	424	360	6.58	3.12
ht_c3_03	28	*	*	275	240	0	275	0	65	515	340	7.21	2.64

A Tabela 3 detalha o tempo gasto na geração de colunas para cada uma das estratégias.

Tabela 3: Tempo gasto na geração de colunas (em segundos)

		Opt?		Tempo Res.			Tempo Comb.			Tempo Total Ger.		Tempo Total	
Nome	Itens	Res.	Comb.	KN	MIP	Comb.	KN	MIP	Comb.	Res.	Comb.	Res.	Comb.
beng01	20	*	*	0.01	249.17	0.00	0.01	157.02	21.85	250.39	179.82	302.18	226.62
beng02	40	*	*	0.04	209.78	0.00	0.04	170.74	30.12	210.37	201.71	244.79	249.15
beng04	80			0.20	3054.17	0.00	0.22	2415.74	401.87	3060.45	2826.58	3600.04	3600.77
beng05	40			0.09	2954.56	0.00	0.10	2192.27	562.80	2965.19	2769.21	3600.18	3600.02
cgcut1	7	*	*	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.04	0.03
cgcut2	10	*	*	0.01	1.38	0.00	0.01	0.00	0.07	1.40	0.08	1.71	0.36
gcut01	10	*	*	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01
gcut02	20	*	*	0.05	0.08	0.00	0.05	0.00	0.03	0.14	0.09	0.22	0.19
gcut03	30	*	*	0.10	0.00	0.00	0.10	0.00	0.00	0.10	0.11	0.18	0.18
gcut05	10	*	*	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.01	0.04	0.04
gcut06	20	*	*	0.09	12.52	0.00	0.08	9.21	7.10	12.63	16.42	13.97	17.81
gcut07	30	*	*	0.16	175.97	0.00	0.18	126.55	86.48	176.27	213.35	185.05	221.38
gcut08	50	*	*	1.41	0.29	0.00	1.49	0.00	0.29	1.70	1.79	2.03	2.15
gcut09	10	*	*	0.01	0.49	0.00	0.02	0.00	0.22	0.51	0.25	0.62	0.39
gcut10	20	*	*	0.11	0.00	0.00	0.12	0.00	0.00	0.11	0.12	0.15	0.16
gcut11	30	*	*	0.50	0.00	0.00	0.52	0.00	0.00	0.50	0.53	0.60	0.64
gcut12	50	*	*	1.23	0.00	0.00	1.29	0.00	0.00	1.23	1.30	1.34	1.40
gcut13	32			6.35	3407.85	0.00	6.72	955.41	2568.58	3417.12	3531.94	3601.07	3606.87
ht_c1_01	16	*	*	0.00	0.15	0.00	0.00	0.00	0.00	0.15	0.01	0.25	0.10
ht_c1_02	17	*	*	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.01	0.11	0.12
ht_c1_03	16	*	*	0.01	0.00	0.00	0.01	0.00	0.00	0.02	0.01	0.16	0.17
ht_c2_01	25	*	*	0.09	0.00	0.00	0.07	0.00	0.00	0.10	0.09	1.26	1.26
ht_c2_02	25	*	*	0.14	0.05	0.00	0.12	0.00	0.00	0.22	0.15	1.86	1.70
ht_c2_03	25	*	*	0.10	0.00	0.00	0.08	0.00	0.00	0.12	0.10	1.42	1.41
ht_c3_01	28	*	*	0.14	114.81	0.00	0.11	0.00	4.97	115.39	5.25	145.77	15.22
ht_c3_02	29	*	*	0.42	2.69	0.00	0.33	0.00	0.06	3.16	0.43	6.58	3.12
ht_c3_03	28	*	*	0.23	2.92	0.00	0.19	0.00	0.10	3.21	0.32	7.21	2.64

É importante observar que, para as instâncias onde o algoritmo com a estratégia de combinações obteve expressivo ganho de tempo, todas as colunas foram geradas utilizando o algoritmo combinatório, com excessão da instância `beng01`.

6. Conclusões

Neste trabalho apresentamos um algoritmo de *branch-and-price* para o PEBF2. Nossa implementação utilizou a regra de ramificação proposta por Vance (1994). Notamos que a geração de colunas com imposição de restrições de aresta é um gargalo de tempo na execução do algoritmo. Por esse motivo desenvolvemos a estratégia de gerar algumas combinações válidas de itens e aplicar sobre estas, o algoritmo de programação dinâmica. Tal estratégia trouxe ganhos significativos de tempo em algumas instâncias. Uma extensão deste trabalho seria utilizar heurísticas na geração de

colunas em nós com restrições de aresta. Apenas no casos onde a coluna gerada pela heurística não tiver custo reduzido negativo, aplicaríamos o resolvedor.

Agradecimentos

Agradecemos o suporte financeiro da Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP, e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES. Este é o projeto de pesquisa PROC. FAPESP NO.04/12711-0 vinculado ao PROJETO TEMÁTICO PRONEX - FAPESP/CNPq PROC. FAPESP NO. 2003/09925-5, intitulado *Fundamentos da Ciência da Computação: Algoritmos Combinatórios e Estruturas Discretas*.
http://pronex-focos.incubadora.fapesp.br/portal.

Referências

BARNHART, C.; JOHNSON, E. L.; NEMHAUSER, G. L.; SAVELSBERGH, M. W. P.; VANCE, P. H. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, INFORMS, v. 46, n. 3, p. 316–329, 1998. ISSN 0030-364X.

BEASLEY, J. E. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, v. 36, p. 297–306, 1985.

BENGTSSON, B. Packing rectangular pieces - A heuristic approach. *The Computing Journal*, v. 25, p. 353–357, 1982.

CHRISTOFIDES, N.; WHITLOCK, C. An algorithm for two-dimensional cutting problems. *Operations Research*, v. 25, p. 30–44, 1977.

COIN. *COmputational INfrastructure for Operations Research*.
http://www.coin-or.org.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. [S.l.]: Freeman, San Francisco, 1979.

GILMORE, P.; GOMORY, R. A linear programming approach to the cutting stock problem. *Operations Research*, v. 9, p. 849–859, 1961.

GILMORE, P.; GOMORY, R. The theory and computation of knapsack functions. *Operations Research*, v. 15, p. 1045–1075, 1967.

HOPPER, E.; TURTON, B. C. H. A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems. *Artificial Intelligence Review*, v. 16, p. 257–300, December 2001.

HOPPER, E.; TURTON, B. C. H. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. *European Journal of Operations Research*, v. 128, p. 34–137, 2001.

KANTOROVIC, L. V. Mathematical methods of organizing and planning prodution. *Management Science*, v. 6, p. 363–422, 1960. (in Russian 1939).

KENYON, C.; RéMILA, E. A Near-optimal solution to two-dimensional cutting stock problem. *Mathematics of Operations Research*, v. 25, p. 645–656, 2000.

LODI, A.; MARTELLO, S.; VIGO, D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, v. 11, n. 4, p. 345–357, 1999.

LODI, A.; MARTELLO, S.; VIGO, D. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, Elsevier Science Publishers B. V., v. 123, n. 1-3, p. 379–396, 2002.

LODI, A.; MARTELLO, S.; VIGO, D. Models and Bounds for Two-Dimensional Level Packing Problems. *Journal of Combinatorial Optimization*, v. 8, p. 363–379, 2004.

MARTELLO, S.; MONACI, M.; VIGO, D. An Exact Approach to the Strip-Packing Problem. *INFORMS Journal on Computing*, v. 15, n. 3, p. 310–319, 2003.

MARTELLO, S.; TOTH, P. *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. ISBN 0-471-92420-2.

RYAN, D. M.; FOSTER, B. A. An integer programming approach to scheduling. In: WREN, A. (Ed.). *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. [S.l.]: North-Holland Publishing Company, 1981. p. 269–280.

VANCE, P. H. Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound. *Computational Optimization and Applications*, v. 3, p. 111–130, 1994.

VANDERBECK, F. On Dantzig-Wolfe Decomposition in Integer Programming and ways to Perform Branching in a Branch-and-Price Algorithm. *Operations Research*, INFORMS, v. 48, n. 1, p. 111–128, 2000. ISSN 0030-364X.

Xpress-MP®. *Dash Optimization*.
`http://www.dashoptimization.com`.