

**Um algoritmo exato para o Problema de
Empacotamento Bidimensional em Faixas**

Carlos Eduardo de Andrade

Dissertação de Mestrado

Um algoritmo exato para o Problema de Empacotamento Bidimensional em Faixas

Carlos Eduardo de Andrade¹

Setembro de 2006

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa (Orientador)
- Prof. Dr. Carlos Eduardo Ferreira
Instituto de Matemática e Estatística - USP
- Prof. Dr. Cid Carvalho de Souza
Instituto de Computação - UNICAMP
- Profa. Dra. Yoshiko Wakabayashi (Suplente)
Instituto de Matemática e Estatística - USP
- Prof. Dr. Orlando Lee (Suplente)
Instituto de Computação - UNICAMP

¹Este projeto tem o apoio da FAPESP, processo No. 04/12711-0, e da CAPES.

IDADE BC
CHAMADA TI UNICAMP
An 24a
EX
MBO BC/ 70740
OC. 16.133-06
D X
EÇO 11.00
TA 28/11/06
3-ID 392133

**FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP**

Bibliotecária: Miriam Cristina Alves – CRB8a / 9054

Andrade, Carlos Eduardo de.

An24u Um algoritmo exato para o problema de empacotamento bidimensional em faixas / Carlos Eduardo de Andrade -- Campinas, [S.P. :s.n.], 2006.

Orientador: Flávio Keidi Miyazawa

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Algoritmos. 3. Programação inteira. 4. Pesquisa operacional. I. Miyazawa, Flávio Keidi. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: A exact algorithm to two-dimensional level strip packing.

Palavras-chave em inglês (Keywords): 1. Combinatorial optimization . 2.Algorithms. 3. Integer programming. 4. Operational research.

Área de concentração: Teoria da computação

Titulação: Mestre em Ciência da Computação

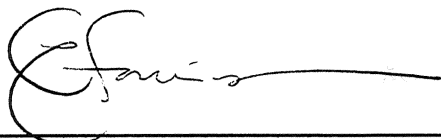
Banca examinadora: Prof. Dr. Flávio Keidi Miyazawa (IC-UNICAMP)
Prof. Dr. Cid Carvalho de Souza (IC-UNICAMP)
Prof. Dr. Carlos Eduardo Ferreira (IME-USP)

Data da defesa: 26/09/2006

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 26 de setembro de 2006, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Carlos Eduardo Ferreira
IME - USP



Prof. Dr. Cid Carvalho de Souza
IC - UNICAMP



Prof. Dr. Flávio Keidi Miyazawa
IC - UNICAMP

Um algoritmo exato para o Problema de Empacotamento Bidimensional em Faixas

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Carlos Eduardo de Andrade e aprovada pela Banca Examinadora.

Campinas, 26 de setembro de 2006.

Prof. Dr. Flávio Keidi Miyazawa
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

© Carlos Eduardo de Andrade, 2006.
Todos os direitos reservados.

Resumo

Problemas de corte e empacotamento aparecem freqüentemente na indústria e comércio, e sua solução de forma otimizada pode trazer grandes ganhos em diversos setores. Um problema muito comum, notadamente no setor têxtil e do papel, é o corte de um rolo ou faixa de um determinado material para obtenção de itens menores, onde temos por objetivo utilizar a menor extensão do rolo/faixa possível. Este problema, conhecido como Problema de Empacotamento Bidimensional em Faixas (PEBF), é tido como um problema de otimização combinatória de difícil resolução. Neste trabalho, apresentamos um algoritmo exato para o PEBF restrito a cortes de dois estágios (PEBF2). O algoritmo usa a técnica de *branch-and-price*, que utiliza, por sua vez, heurísticas baseadas em algoritmos aproximados para a obtenção de limitantes superiores. O algoritmo se mostrou eficaz na obtenção de soluções para instâncias de pequeno e médio porte.

Abstract

Cutting and packing problems are common problems that occur in many industry and business process. Their optimized resolution leads to great profits in several sectors. A common problem, that occur in textil and paper industries, is to cut a strip of some material to obtain several small items, using the minimum length of material. This problem, known by Two Dimensional Strip Packing Problem (2SP), is a hard combinatorial optimization problem. In this work, we present an exact algorithm to 2SP, restricted to two staged cuts (known by Two Dimensional Level Strip Packing, 2LSP). The algorithm uses the *branch-and-price* technique, and heuristics based on approximation algorithms to obtain upper bounds. The algorithm obtained optimal or almost optimal for small and moderate sized instances.

*A meus pais José Carlos e Nilza.
A meus irmãos Bruno e Isis.
E a todos que acreditam em seus sonhos.*

Agradecimentos

Um de meus amigos tem uma teoria interessante: toda leitura começa pelos agradecimentos. Eu digo mais, talvez nem passe desse ponto (tomara que esse não seja seu caso, leitor!!!). Acredito que isso seja verdade porque esta seção é o local onde o autor pode se expressar livremente, sem algum rigor estabelecido. Mas isso não é inteiramente verdade pois são nos agradecimentos que os créditos devem ser dados, e se segue um certo rito para tal. De qualquer maneira, considero de importância única essa seção, pois ela é capaz de mostrar algumas influências na vida do autor e suas contribuições para formação deste. A seção de agradecimentos mostra a pessoa por trás do autor (científico). Sendo assim, chega de enrolação e vamos ao rito. Para ser justo, seguirei colocando os nomes em ordem alfabética em cada bloco.

Como de praxe, a família vem em primeiro lugar. E nada mais justo que isso, pois minha família sempre foi e será a base de minha vida e formação. Agradeço e louvo meus pais José Carlos e Nilza pelos incessantes amor, carinho, dedicação, paciência e educação, voltados para meus irmãos e eu.

Obrigado, meu pai, pelo constante estímulo, desde pequenino, a sempre buscar pela verdade e sabedoria, sem dogmas e preconceitos. Obrigado por ensinar-me a trilhar meu caminho e acreditar em meus sonhos (“Por que você quer ser igual aos outros, meu filho? Não, você deve ser diferente, somente assim vai crescer”, nos meus 6 anos de idade; “Pai, quero ser um inventor!”, “Inventor, filho? Quer dizer, cientista?”, “Isso pai, cientista! Que preciso fazer?”, “Tem que estudar, meu filho, tem que estudar...”, também nos meus 6 anos. Jamais esquecerei destas palavras). Obrigado minha mãe por me ensinar a partilhar, a trabalhar e a ter bom humor. Obrigado pelo carinho, amor e compreensão, sempre reconfortantes. Obrigado por sempre acreditar em mim, sempre me proteger mas nunca em demasia. Pelas imensas lições de partilha e compreensão. Obrigado aos dois por me ensinarem a buscar e lapidar o que é bom, belo e justo, *SEMPRE!*

Agradeço a meus irmãos Bruno e Isis pela paciência, brigas, festas e travessuras. Nossa pouca diferença de idade sempre foi favorável para nosso crescimento em conjunto. Obrigado Bruno pelas caronas de carro e sua alma prestativa. Obrigado Isis

pelos doces e bolos, por dar uma olhada no português desta dissertação e pela sempre prontidão. Obrigado as dois pelas conversas fiadas, momentos de lazer e ajuda mútua. E tia Neide e tio Kleber que sempre me apoiaram e ajudaram.

Aproveitando que estou em Ouro Fino, agradeço a todos meus amigos de lá. Em especial aos sempre companheiros Odair e Richard, cuja amizade perdura por mais de 14 anos. Lá por perto, na Escola Agrotécnica Federal de Inconfidentes, tive ótimos professores, pessoas cuja minha gratidão é eterna. Em especial, minha prof. Mônica Bartholo e seu marido prof. Fernando Bartholo, muito mais que professores, verdadeiros amigos para vida. Obrigado "Monca"!

Indo para frente no tempo, agradeço a amizade fiel e sincera de Flávio L. N. Batista e Rodrigo F. Toso (Podrão para os íntimos), grandes amigos, colegas de graduação, e irmãos de combate. Mesmo distantes, sempre nos mantivemos ligados. Aos meninos Todé (André R. Barros) e Rafael (M. D. Frinhani) sempre bons amigos. Este último, por ser o irmão mais velho que não tive.

Vindo para Campinas, a lista cresce. Primeiro, agradeço meu orientador Flávio, que me acolheu no IC e acreditou em meu trabalho. E mais do que isso, proporcionou-me um crescimento científico sem tamanho. Foi paciente e compreensivo, mesmo nas minhas dúvidas e/ou erros mais banais. Sempre esteve pronto a me atender e ajudar. Sou muito grato mesmo! Agradeço ao prof. Cid C. de Souza pela amizade, disposição e as aulas de programação inteira. Você está na lista de meus melhores professores.

A UNICAMP me deu grandes amigos. Seria injusto não falar de cada um deles.

Agradeço a meu grande amigo Diego S. T. Martinez, pelo sempre companheirismo e preocupação. Embora tenha reclamado, às vezes, das longas conversas madrugada adentro sobre ciência e educação, elas foram cruciais na minha formação. Obrigado pelas cachaçadas e devaneios. Obrigado por sempre estar ao meu lado quando precisei. Saiba que tem meu profundo respeito e admiração.

Outra pessoal que tem meu profundo respeito e admiração é Rafael F. dos Santos (vulgo Mamão), companheiro de laboratório, república e sofrimento. Obrigado pela paciência com meu ser falante, quase esquizofrênico, e meus chilikques corriqueiros. Tenho você como grande amigo e irmão.

Agradeço a meus irmãos acadêmicos Eduardo (vulgo Pará), Luiz Meira, Evandro Branch e André Vignatti (vulgo Guri), companheiros na dolorida vida de teórico da computação. Em especial, agradeço a Pará pela nem sempre paciência com minhas manias e loucuras, mas sempre paciência com minhas dúvidas com esse projeto. E a André, o segundo cara mais camarão do IC (vide agradecimentos em Vignatti [64]), pela paciência com minhas alegorias científicas (as minhas famosas analogias), pelas boas músicas, pela cachaçada e pelas boas risadas.

Ao pessoal da moradia da UNICAMP. Em especial, Marcelo (Y. Cunita) pelas aulas

de história, pelos bons filmes e livros, pela receita do yakissoba e por me ensinar a fazer os tsurus. Rodrigo Dutra, vulgo Roy, pelas deliciosas viagens imaginárias aos confins da mente, cerne da filosofia e ciência: como é difícil achar pessoas assim. Mas ainda assim, achei um pequeno sol, que vem iluminando minha vida nesses últimos meses. O nome dessa estrela anã-branca é Andréia F. de Faria, que tem se mostrado grande companheira até então. Mulheres inteligentes, quem pode com elas?

Aos Malditos (calma, é só o nome da república), “zomi ruim” (plural de “homi ruim”) Celsinho, Didjei (vulgo Richard), Flavinho, Gustavo, Mamão (de novo!), Mário (que Mário?!?), Pagodinho (vulgo Dárcio), Paixão (vulgo Márcio), e Napoleão (vulgo Polli). Embora este último seja um canino, sempre se mostrou grande amigo, mesmo de assaltantes de república!!!

Ao pessoal do Wushu/Sanshou, em especial, Gabriel, Júlia (ambas), Leonardo, Marcos (esse é o cara!), Mariana, Pedro, Priscila, Raquel e Tomaz. E as eternas frases “No pain, no gain!!!”, “Quando vou aprender a voar?!?!”. Acho que hoje compreendo o verdadeiro significado de masoquismo: acordar todo dia às 6hs da matina, faça sol, chuva ou neve, e ir martirizar-se, ficando dolorido o dia todo.

A todo pessoal do IC, em especial ao falecido LAD, por vezes LUG, agora definitivamente LOCO (Laboratório de Otimização Combinatória), André Ciré, Daniel Ferber, Guri, Mamão, Nilton, Pará, Peterson, Tony e Vitor. Ao pessoal da pós: Adilson, André Atanásio, Bruno, Borin, Celso, Cláudio, Daniel Macedo, Edna, Evandro Bacarin, Fernanda, Juliana Borin, Juliana de Santi, Leo B., Leo BSD, Leonel, Luiz Bit., Neumar, Sheila, Patric, Tiago Moronte e Thiago Coelho. Também agradeço aos funcionários do IC, em especial, Seu Nelson, pelo bom humor de todos os dias, e dona Neuza pelo delicioso café (combustível de computeiro). Se esqueci alguém e isso configurar uma injustiça, desculpe-me. Fique a vontade para colocar seu nome no final, ou em lugar que melhor achar!

Destes últimos, mais em especial, aos que participavam dos almoços filoso-científico-religioso-cético-culturais no bandeirão, onde quase ninguém contava algum “acontecido”, emitida sua polêmica opinião mais que pessoal ou embarcava em discussões que duravam dias, às vezes, semanas. Lá nasceram muitas teorias, inclusive sobre o *motor de reversão*, incrível dispositivo motriz capaz de modificar seu sentido de rotação sem apresentar grande consumo de energia. Mas isso é assunto para outra tese...

Este parágrafo nunca será lido (bom, acredito eu nisso) pelos nomes que citarei. Mas ainda assim, gostaria de deixar minhas lembranças a estes que fizeram parte de minha formação. Acho que isso fica como dica para você, leitor. Agradeço aos pensadores Sócrates, Platão, Aristóteles e Nietzsche, aos cientistas (também pensadores, óbvio) Newton, Poncairé, Bohr e cia, Einstein, Turing, von Neuman; aos filósofos da ciência (também cientistas e pensadores) Richard Dawkins e Carl Sagan. Aos escritores

Gabriel Garcia Marques (por Cem Anos de Solidão e outros) e Douglas Adams (pela série Guia do Mochileiro das Galáxias). Aos diretores Stanley Kubrick (pelos excelentes Laranja Mecânica, Nascido para matar e De Olhos Bem Fechados), Quentin Tarantino (por Pulp Fiction e Kill Bill), Alfred Hitchcock (por Os pássaros e Janela Indiscreta), Peter Jackson (pelos filmes da trilogia Senhor dos Anéis), Zhang Yimou (pelo filme Herói), e Roberto Benigni (pela atuação no belíssimo filme “A Vida é Bela”). Ao pessoal do Pink Floyd, Jethro Tull, Neil Young e cia., Dream Theater, Oasis, Rappa, Paralamas do Sucesso, Skank, Engenheiros do Hawaii e Almir Sater. O pessoal da Pixar pelas animações Monstros S.A. e Procurando Nemo, o pessoal da Blue Sky por Era do Gelo 1 e 2, o pessoal da Dreamworks por Shrek 1 e 2, e o pessoal da Disney por Irmão Urso. Matt Groening por The Simpsons, a melhor série de desenhos animados!!! O pessoal da Nintendo, Square e Capcom, por seus excelentes jogos, e o pessoal da Infinity Ward e Activision pelo jogo Call of Duty. Ao Batman e seus inimigos Pingüim e Coringa. Existem muitos outros desenhos, animações, filmes e jogos que estão em minha lista, mas não dá para colocar aqui. Todos me inspiram profundamente, me confortam e me fazem sonhar. E isso é muito importante pois **acredito que os sonhos são os motores do mundo**. *Simples, eterna e etérea é minha alma. Etérea e infinita poesia da vida. Sou etéreo então. E eternos são meus sonhos. Voe ao longe, pegue um raio de sol e traga-o a mim.*

P.S.: Ah, agradeço, e muito mesmo, a FAPESP e a CAPES pelo apoio financeiro de meu projeto. Sem este apoio, nada disso seria possível!!! E claro, a toda judiada população brasileira que financia estas entidades, escolas e universidades públicas. É o que eu acredito: o Brasil só anda com educação!!!

*As mentes humanas são treinadas largamente às custas de seus corações.
Não é apenas isto; é apenas a existência de mais mentes educáveis que corações educáveis.*
(Marie von Eschenbach)

*Aprender é como remar contra a correnteza,
sempre que se pára, anda-se para trás.*
(Confúcio)

Sumário

Resumo	vii
Abstract	viii
Agradecimentos	x
1 Introdução	1
1.1 Objetivos	3
1.2 Organização do Texto	3
2 Problemas de Corte e Empacotamento	5
2.1 Introdução	5
2.2 Estrutura dos Problemas de Corte e Empacotamento	5
2.3 Problemas Unidimensionais	9
2.4 Problemas Bidimensionais	10
2.5 Problemas com mais de 2 dimensões e variações no empacotamento . .	13
3 Técnicas de Resolução	16
3.1 Algoritmos Aproximados	16
3.2 Programação Dinâmica	17
3.3 Programação Linear Inteira	18
3.4 Geração de Colunas	21
3.4.1 Introdução	21
3.4.2 A Decomposição de Dantzig-Wolfe	23
3.4.3 Geração de colunas para problemas de corte e empacotamento .	27
3.5 O Método Branch-and-Bound	28
3.5.1 Introdução	28
3.5.2 O algoritmo de Branch-and-Bound e a Programação Linear . . .	31
3.6 O Método Branch-and-Price	31
3.6.1 Considerações sobre algoritmos de Branch-and-Price	33

4	Algoritmos Aproximados e Heurísticas para o PEBF2	35
4.1	O algoritmo Next-Fit Decreasing Height	35
4.2	O algoritmo First-Fit Decreasing Height	37
4.3	O algoritmo Best-Fit Decreasing Height	38
4.4	O algoritmo da Mochila Gulosa	39
4.5	Outras abordagens	41
5	Método Branch-and-Price para o PEBF2	42
5.1	Formulações	42
5.2	O algoritmo	45
5.2.1	Ramificação	45
5.2.2	Geração de Colunas	48
5.2.3	Limitantes	50
5.3	Implementação	51
6	Experimentos Computacionais	53
6.1	Ambiente Computacional	53
6.2	Instâncias Testadas	53
6.3	Verificação do algoritmo	54
6.4	Limitantes Superiores	55
6.5	Resultados Iniciais	60
6.6	Geração de Colunas	64
6.7	Resultados Finais	67
7	Conclusão e Considerações Finais	74
	Bibliografia	76

Lista de Tabelas

6.1	Resultados das instâncias do PEU: características da árvore	54
6.2	Resultados das instâncias do PEU: geração de colunas	55
6.3	Comparação dos algoritmos para geração do limitante superior	56
6.4	<i>Gap</i> entre a solução ótima e a solução de cada algoritmo	59
6.5	Resultados: mochila gulosa e BFDH: características da árvore	62
6.6	Resultados: mochila gulosa e BFDH: geração de colunas	63
6.7	Comparação da geração de colunas pelo resolvidor e por combinações .	65
6.8	Comparação geração de colunas: ramificação simples	66
6.9	Comparação geração de colunas: ramificação com itens diferentes	66
6.10	Comp. entre num. de itens nas restrições de aresta: quant. de variáveis .	67
6.11	Comp. entre num. de itens nas restrições de aresta: tempos	67
6.12	Resultados: 1ª var. com CR negativo: características da árvore	69
6.13	Resultados: 1ª var. com CR negativo: geração de colunas	70
6.14	Comparação entre configurações básica e PCRN do algoritmo	71

Lista de Figuras

2.1	Tipos básicos de problemas de corte e empacotamento	8
2.2	Exemplo de corte de canos interpretado como PCU	11
2.3	Exemplos de empacotamento ortogonais e não-ortogonais	11
2.4	Exemplo de corte de um rolo de papel	13
2.5	Padrões de corte/empacotamento	14
3.1	Esquema de geração de colunas para um problema de minimização . . .	23
3.2	Método de <i>Branch-and-Bound</i> para problemas de minimização	29
3.3	Esquema do método de <i>Branch-and-Price</i> para problemas de minimização	32
4.1	Empacotamento utilizando NFDH	36
4.2	Empacotamento utilizando FFDH	38
4.3	Empacotamento utilizando BFDH	39
6.1	Limitantes para instâncias de beng01 a beng10	58
6.2	Limitantes para instâncias de gcut01 a gcut12	60
6.3	Árvore de ramificação da instância cgcut2	73

Lista de Algoritmos

1	Programação dinâmica para o Problema da Mochila	19
2	<i>Next-Fit Decreasing Height</i>	36
3	<i>First-Fit Decreasing Height</i>	37
4	<i>Best-Fit Decreasing Height</i>	38
5	Mochila Gulosa	40

Capítulo 1

Introdução

Desde o início da revolução industrial, as operações econômicas dos produtores de bens apresentam uma série de complicações, geralmente ligadas a tomadas de decisões vitais à sobrevivência destes produtores no mercado. Muitas decisões são relativas à produção e logística destes bens e pequenas variações no custo destas operações podem trazer grandes ganhos, o que, por sua vez, podem favorecer a competitividade das empresas no mercado.

Dentre uma imensa variedade de problemas industriais, como transporte de produtos, seqüenciamento de máquinas em linhas de produção, entre outros, temos os problemas de corte e empacotamento. Problemas de empacotamento, em geral, são aqueles que requerem que certos objetos, chamados de itens, sejam empacotados em outros de tamanhos maiores, chamados de recipientes. Em algumas aplicações, ao invés de empacotar, o objetivo é cortar. Mas, em ambos problemas, os itens devem ser empacotados ou cortados sem sobreposição.

De fato, os problemas de corte e empacotamento são equivalentes pois representam a mesma tarefa no espaço, ou seja, o ato de empacotar ou cortar remete à divisão de um espaço (seja qual for a dimensão) em partições onde serão alocados os itens a serem empacotados/cortados. Imagine, por exemplo, a carga de um caminhão onde os itens não podem ser colocados um em cima do outro. Se temos um grande número de itens, vários caminhões serão necessários para transportá-los. Se cada caminhão representa um custo (frete ou combustível, por exemplo) precisamos acomodar os itens de tal maneira que um número mínimo de caminhões seja necessário. Por outro lado, imaginemos uma vidraçaria contratada para uma grande obra. Vários tipos e formatos de vidros são requeridos. Como, geralmente, as vidraçarias trabalham com grandes placas de vidro, estas devem ser cortadas para obtenção de pedaços de vidros menores, de modo que minimizemos as perdas ou o número de placas necessárias. Em ambos casos, a alocação ou corte de itens se dão em duas dimensões e um item não pode ocupar

a área de outro. Embora estes problemas sejam equivalentes, as diferenças ocorrem quando consideramos as diferentes restrições que ocorrem em cada caso, como, por exemplo, tipo de corte ou tipo de itens a serem empacotados juntos.

Os 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 à criação de peças de roupa, ou, o corte de barras de aço para atender 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, papel 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* PEBF (do inglês *Two Dimensional Strip Packing*).

No geral, as máquinas utilizadas para o corte desses materiais operam apenas no 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 alternados sucessivamente no sentido horizontal e vertical. Uma restrição comum é uma limitação no número de estágios de cortes para obter-se os itens finais. Tais restrições são comuns quando o custo envolvido nos cortes são relativamente altos.

Podemos notar que existem muitas maneiras de como cortar ou empacotar itens dentro de recipientes. Por isso, os problemas de corte e empacotamento são considerados problemas de otimização combinatória e trazem consigo muitas variantes como: tipos de cortes, dimensionalidade, metas a serem alcançadas, geração de resíduos, entre outros, que influem substancialmente na maneira de construir as combinações de itens/recipientes.

Um problema de otimização combinatória pode ser definido como ação de maximizar ou minimizar uma função de várias variáveis sujeita a um conjunto de restrições, dentro de um domínio finito e enumerável. Embora o problema tenha um domínio finito, geralmente ele é muito grande e a simples enumeração das soluções pode tomar unidades muito grandes inviabilizando seu tratamento computacional.

Em vista disto, métodos que cubram eficientemente o espaço de busca são muito estudados. A variedade destes algoritmos é grande, passando por métodos exatos, heurísticos e de aproximação. Destes, destacam-se os métodos exatos de resolução, por apresentarem soluções ótimas dos problemas a que se destinam, e os métodos de aproximação que garantem uma qualidade da solução.

Entre os métodos exatos, destacam-se os métodos de *branch-and-bound* que utilizam técnicas baseadas na qualidade dos limitantes do problema tratado. Os limitantes

permitem a exclusão de várias soluções (mais importante, conjuntos de soluções cuja estrutura seja parecida) reduzindo o espaço de busca por uma solução ótima. Estes limitantes podem ser calculados através de programação linear. Dentre as variantes do método de *branch-and-bound* com programação linear, destaca-se o método de *branch-and-price* que permite a utilização de formulações de problemas cujo número de variáveis é muito grande. Em geral, tais formulações são mais restritivas e apresentam limitantes de boa qualidade.

1.1 Objetivos

Este trabalho tem por objetivo principal investigar a resolução de forma exata do problema de empacotamento bidimensional em faixas com 2 estágios (PEBF2). Para tanto, procuramos:

- investigar e implementar métodos para obtenção de bons limitantes primais e duais;
- investigar e implementar métodos para geração de colunas, e
- desenvolver um algoritmo de *branch-and-price*.

A escolha do PEBF2 justifica-se por ser um problema muito comum na indústria e de importância única nos setores têxtil, metalúrgico, do papel e celulose, moveleiro (indústria de móveis), entre outros.

Devido aos recentes avanços no hardware, podemos utilizar uma abordagem exata que garante que a solução encontrada é a melhor possível. Assim, utilizamos a técnica de *branch-and-price*, que garante bons limitantes e soluções em tempo satisfatório.

O trabalho justifica-se academicamente pois, embora existam algumas heurísticas e algoritmos aproximados para o PEBF2, este problema não foi tratado, até onde sabemos, através do método de *branch-and-price*. Lodi *et al.* [43] implementaram algoritmos para geração de limitantes duais e compararam seus resultados com os resultados da relaxação linear. Utilizaram ainda, o algoritmo de *branch-and-bound* de um resolvidor comercial para resolver o PEBF2 mas não levaram em conta nenhuma especificidade (formulação, geração de colunas ou regras de ramificação customizadas).

1.2 Organização do Texto

O Capítulo 2 apresenta os principais problemas de corte e empacotamento, sua tipologia e variedade. Este capítulo considera as principais variantes e exemplificações das

mesmas.

O Capítulo 3 apresenta diversas técnicas de resolução que podem ser aplicadas a estes problemas. Especificamente, as técnicas apresentadas foram utilizadas na construção do algoritmo proposto no Capítulo 5. São apresentados os métodos aproximados, programação dinâmica, programação linear inteira e métodos de *branch-and-bound*.

O Capítulo 4 apresenta dois algoritmos aproximados e duas heurísticas o PEBF2, uma delas baseada em programação dinâmica. Estes algoritmos são utilizados para os cálculos dos limitantes superiores do problema.

O Capítulo 5 mostra formulações, compacta e exponencial, do PEBF2, propondo um algoritmo de *branch-and-price* baseado nesta última. Neste capítulo, são tratados os problemas com a ramificação e geração de colunas da formulação utilizada.

O Capítulo 6 apresenta os experimentos computacionais dos algoritmos implementados e o Capítulo 7 traz as considerações finais deste trabalho e futuros possíveis desenvolvimentos e/ou abordagens.

Capítulo 2

Problemas de Corte e Empacotamento

2.1 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. Embora estes problemas sejam equivalentes, as diferenças ocorrem quando consideramos as diferentes restrições que ocorrem em cada caso.

Os problemas de corte e empacotamento são amplamente estudados devido ao grande interesse prático e teórico. Algumas aplicações reais ocorrem em problemas de corte de placas/vidros/isopor, empacotamento em contêineres, inserção de comerciais em TV, escalonamento de processos, etc. Dyckhoff *et al.* [21] apresentam uma extensa bibliografia anotada. Outros *surveys* podem ser encontrados nos trabalhos de Dyckhoff e Finke [20] e Dowsland e Dowsland [18]. Lodi *et al.* [40] apresentam um *survey* de problemas bidimensionais.

2.2 Estrutura dos Problemas de Corte e Empacotamento

Os problemas de corte e empacotamento têm uma estrutura idêntica e podem ser descritos de forma similar. De forma geral, eles têm dois conjuntos de objetos:

- um conjunto de objetos de grande tamanho, chamados *recipientes* e
- um conjunto de objetos pequenos (menores que os recipientes) chamados de *itens*.

Estes conjuntos podem estar definidos em uma, duas, três ou mais dimensões geométricas. Os itens são selecionados e agrupados em conjuntos que são atribuídos aos recipientes de modo que

- todos os itens de um conjunto cabem inteiramente no recipiente a qual foram atribuídos e
- nenhum item pode se sobrepor com outro.

Junto desta estrutura, uma função objetivo deve ser otimizada, como por exemplo, menor número de recipientes usados ou menor perda de material.

Essa caracterização serve de base para todos problemas de corte e empacotamento. A diferença entre os problemas reside em propriedades adicionais exigidas em cada tipo de problema. Segundo Wäscher *et al.* [65], cinco subproblemas podem ser distinguidos:

1. Problema de seleção a respeito dos recipientes;
2. Problema de seleção a respeito dos itens;
3. Problema de agrupamento a respeito dos itens selecionados;
4. Problema de alocação a respeito da atribuição dos conjuntos de itens aos recipientes;
5. Problema de *layout* a respeito da disposição dos itens nos recipientes com respeito a condições geométricas.

O primeiro item diz respeito diretamente aos recipientes utilizados, por exemplo, determinados tipos de recipientes podem ser mais caros em sua produção mas geram menor desperdício, portanto podem ter preferência no corte. O segundo item trata dos itens, onde alguns destes podem ter prioridade de produção, por exemplo. Alguns itens não podem ser empacotados juntos, como produtos químicos e produtos alimentícios: o terceiro item cobre estas considerações. Determinados itens só podem ser alocados em locais específicos, como carga de caminhões, ou corte de itens de diversos materiais: o quarto item considera estas observações. Por fim, o quinto item considera restrições como tipo de corte, por exemplo, degradê de densidade de chapas metálicas e disposição de microchips em circuitos integrados.

Dyckhoff [19] apresentou uma tipologia para problemas de corte e empacotamento, que se mostrou mais tarde incompleta na descrição dos problemas. Essa tipologia é baseada em quatro itens:

1. Dimensionalidade:
 - (1) uma dimensão;
 - (2) duas dimensões;
 - (3) três dimensões;
 - (N) N dimensões, com $N > 3$;
2. Tipo de atribuição:
 - (B) todos recipientes e uma seleção de itens;
 - (V) todos itens e uma seleção de recipientes;
3. Tipo dos recipientes:
 - (O) um recipiente;
 - (I) figuras idênticas;
 - (D) figuras diferentes;
4. Tipo dos itens:
 - (F) poucos itens (de diferentes formas);
 - (M) muitos itens com muitas formas diferentes;
 - (R) muitos itens com poucas formas diferentes;
 - (C) formas iguais.

Esta classificação apresenta algumas falhas, segundo Wäscher *et al.* [65]. Estes propuseram extensões da tipologia de Dyckhoff detalhando de maneira mais precisa as nuances de cada tipo de problema. A classificação básica pode ser vista na Figura 2.1. Wäscher *et al.* propuseram ainda um refinamento destas categorias básicas, descritas em seu trabalho.

Além desta classificação, é usual identificar um problema de corte e empacotamento através das restrições envolvidas. Em geral, nos problemas de corte os itens apresentam demandas, ou seja, vários itens de um determinado tipo podem ser requeridos, enquanto que nos problemas de empacotamento estas demandas são unitárias. Uma restrição comum a problemas de empacotamento é quanto a disposição dos itens, como por exemplo, a estabilidade. Nos problemas de corte, geralmente as restrições envolvidas referem-se ao tipo de corte que é feito ou ao desperdício de material dos recipientes. Entretanto é possível que várias destas características se sobreponham em um determinado tipo de problema, tornando-o híbrido. O trabalho de Wäscher *et al.* [65] traz a caracterização de vários destes problemas e algumas de suas sobreposições.

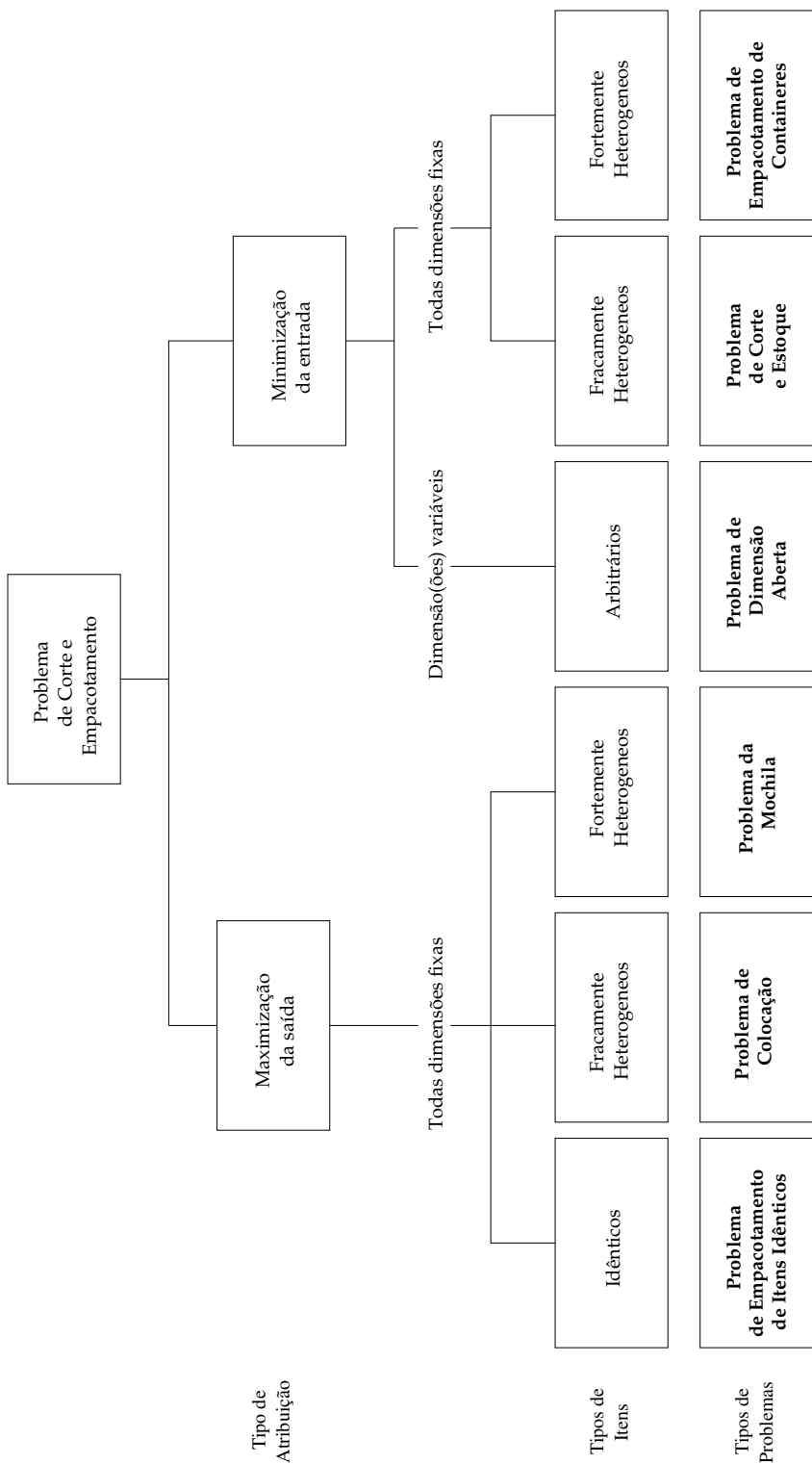


Figura 2.1: Tipos básicos de problemas de corte e empacotamento.

2.3 Problemas Unidimensionais

Um problema é dito ser *unidimensional* quando apenas uma dimensão é considerada no processo. Um dos primeiros e mais famosos problemas de corte e empacotamento unidimensional é conhecido como Problema da Mochila (do inglês *Knapsack Problem*). Uma interpretação pode ser a seguinte: imagine que um vendedor ambulante precisa levar mercadorias em sua mochila. O problema é que a mochila tem uma capacidade restrita e não consegue acomodar todos os itens. O vendedor quer preencher sua mochila de maneira que a soma dos valores dos itens seja a maior possível. Outro exemplo: um investidor tem um certo capital e quer investir em ações. Cada ação tem um valor para compra e um índice de rentabilidade. O objetivo do investidor é maximizar a soma dos índices de rentabilidade das ações que comprar, restrito ao capital que tem para investir.

Sua definição formal é:

PROBLEMA DA MOCHILA

Seja um recipiente de capacidade C e uma lista de itens unidimensionais $I = (r_1, \dots, r_n)$ tal que cada item r_i tem um peso associado $p_i \in (0, C]$ e um valor associado v_i . O objetivo é empacotar alguns itens de I no recipiente de tal maneira que a soma dos pesos dos itens empacotados seja no máximo C e a soma dos valores seja a maior possível.

Este problema também é conhecido como Problema da Mochila Binário ou 0-1 e é NP-difícil, veja Garey e Johnson [26], portanto não pode ser resolvido em tempo polinomial a não ser que $P = NP$. O problema da mochila tem grande importância pois aparece como subproblema de muitos outros problemas mais complexos, e pode representar muitas situações práticas, como nos dois exemplos citados. Por isso, é considerado um dos mais importantes problemas de empacotamento. Sua formulação, em programação linear inteira, pode ser

$$\begin{aligned} & \text{maximize} && \sum_{i \in I} v_i x_i \\ & \text{sob as restrições} && \sum_{i \in I} p_i x_i \leq C \\ & && x_i \in \{0, 1\} \quad \forall i \in I, \end{aligned} \tag{2.1}$$

onde x é um vetor binário de tamanho $|I|$ e cada posição i indica se o item r_i está ou não na solução. A restrição desta formulação, que envolve a capacidade do recipiente, aparece em vários problemas de corte e empacotamento, mesmo que de uma forma ligeiramente alterada. Nas formulações do Capítulo 5 podemos observar que esta restrição aparece com frequência.

Outro problema muito comum é o de empacotar uma lista de itens em vários recipientes. Este problema é conhecido como Problema de Empacotamento Unidimensional (*Bin Packing Problem*):

PROBLEMA DE EMPACOTAMENTO UNIDIMENSIONAL - PEU

Dados recipientes de capacidade C e uma lista de itens unidimensionais $I = (r_1, \dots, r_n)$, tal que cada item r_i tem um peso associado $p_i \in (0, C]$, o objetivo é empacotar os itens de I dentro do menor número de recipientes, de modo que a soma dos pesos dos itens em cada recipiente não exceda sua capacidade.

O PEU também é NP-difícil, veja Garey e Johnson [26]. Uma variante do PEU é quando os itens têm demandas $d_i \in \mathbb{Z}^+$ associadas. O problema é conhecido como Problema de Corte Unidimensional (*Cutting Stock Problem*).

PROBLEMA DE CORTE UNIDIMENSIONAL - PCU

Dados recipientes de capacidade C e uma lista de itens unidimensionais $I = (r_1, \dots, r_n)$ tal que cada item r_i tem um peso associado $p_i \in (0, C]$ e uma demanda $d_i \in \mathbb{Z}^+$, o objetivo é empacotar toda demanda de itens, dentro do menor número de recipientes.

Apesar deste problema generalizar o PEU, não se sabe se a versão de decisão do PCU é NP-completa. Só se sabe que esta se encontra em EXPSPACE^1 . Note que são problemas similares mas têm uma classificação diferente. Em verdade, quando temos demandas de itens tratamos os problemas como problemas de corte. Quando os itens são unitários, tratamos como problemas de empacotamento. A Figura 2.2 mostra um exemplo onde temos demandas de vários tamanhos de canos e nosso material de consumo tem tamanho fixo. Podemos minimizar a utilização de material de consumo, ou o desperdício no corte.

Existe uma infinidade de variações de problemas unidimensionais. Destas, o problema da mochila, o PEU e o PCU são os mais estudados, uma vez que estes se apresentam como subproblemas em diversos outros problemas.

2.4 Problemas Bidimensionais

Nos problemas bidimensionais, duas dimensões são consideradas na resolução. Por exemplo, imagine que uma vidraçaria receba uma grande encomenda de pequenas placas de vidro de vários tamanhos. A vidraçaria trabalha com grandes placas como material de consumo. O objetivo é cortar estas grandes placas nas placas menores, de modo que se utilize ou desperdice a menor quantidade possível de material.

¹EXPSPACE é o conjunto de todos os problemas de decisão que podem ser resolvidos por uma máquina de Turing determinística em $O(2^{p(n)})$, onde $p(n)$ é uma função polinomial em n .

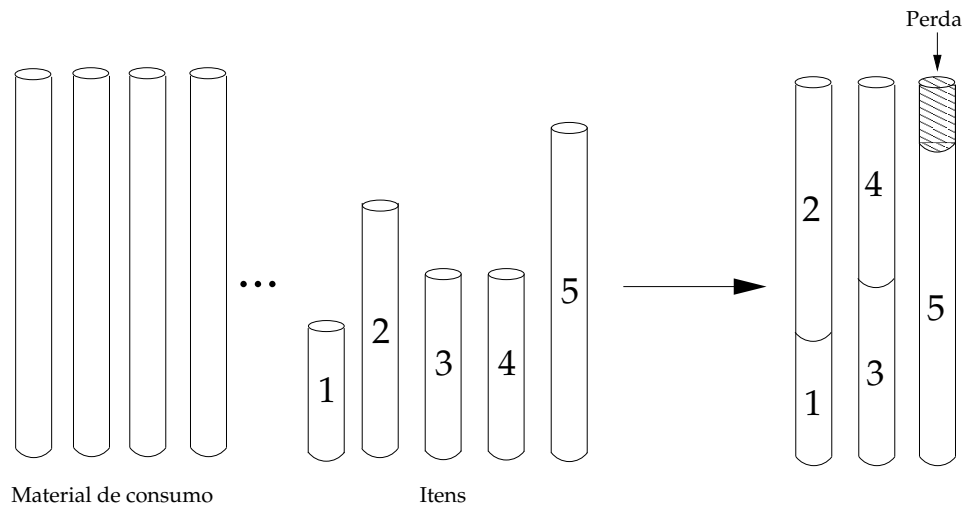


Figura 2.2: Exemplo de corte de canos interpretado como PCU.

Os problemas com duas ou mais dimensões apresentam vários tipos de empacotamento. A Figura 2.3 mostra dois exemplos comuns de empacotamentos. O empacotamento (a), embora os itens sejam figuras regulares, seus lados não estão alinhados com as margens do recipiente. O empacotamento (b) os lados dos itens estão alinhados. Este último tipo é dito ser *ortogonal*, ou seja, os lados dos itens são paralelos ou ortogonais aos lados dos recipientes.

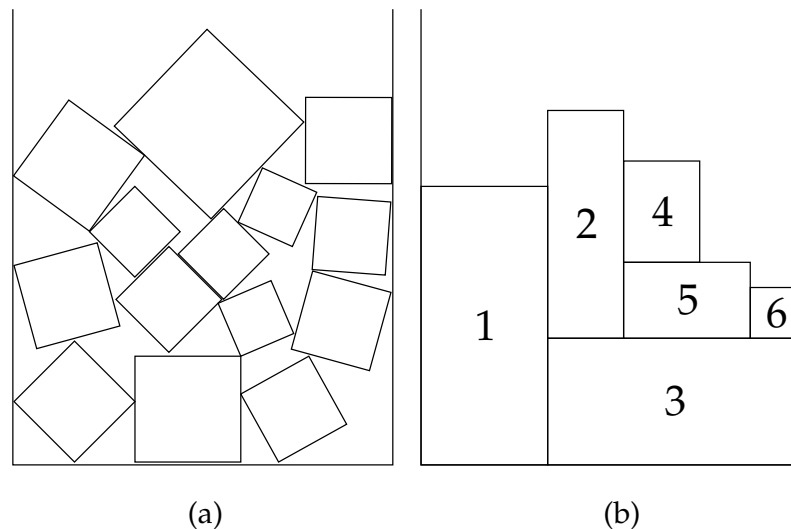


Figura 2.3: Exemplos de empacotamento: (a) não-ortogonal; (b) ortogonal.

Em geral, os problemas que exigem ortogonalidade são mais comuns quando os itens e recipientes são figuras retangulares, já que o esforço para resolução destes pro-

blemas é menor. Neste trabalho, consideramos apenas empacotamentos ortogonais. Todos problemas descritos, salvo referência explícita, são tidos em suas versões ortogonais.

Um problema bem conhecido é o Problema de Empacotamento Bidimensional (*Two Dimensional Bin Packing Problem*), cujo exemplo pode ser visto no início desta seção.

PROBLEMA DE EMPACOTAMENTO BIDIMENSIONAL - PEB

Dados placas bidimensionais $R = (L, A)$ e uma lista de retângulos $I = (r_1, \dots, r_n)$, onde cada retângulo $r_i = (l_i, a_i)$ é tal que $l_i \in (0, L]$ e $a_i \in (0, A]$, para $i = 1, \dots, n$, o objetivo é empacotar os retângulos de I dentro do menor número de placas R .

Claramente o PEB é NP-difícil, já que o PEU se reduz facilmente a ele. Basta adicionarmos mais uma dimensão de tamanho fixo e igual em todos recipientes e itens.

Assim como sua versão unidimensional, o PEB tem uma variação onde os itens têm demandas $d_i \in \mathbb{Z}^+$ associadas. Este problema é conhecido como Problema de Corte Bidimensional em Placas (*Two Dimensional Cutting Stock Problem*).

PROBLEMA DE CORTE BIDIMENSIONAL EM PLACAS - PCB

Dados placas bidimensionais $R = (L, A)$ e uma lista de retângulos $I = (r_1, \dots, r_n)$, onde cada retângulo $r_i = (l_i, a_i)$ é tal que $l_i \in (0, L]$ e $a_i \in (0, A]$, e tem uma demanda $d_i \in \mathbb{Z}^+$, para $i \in I$, o objetivo é empacotar toda demanda de itens dentro do menor número de placas R .

O PCB é um problema muito comum na indústria. Imagine grandes placas de vidro ou metal que devem ser cortadas em itens menores, para atender a demanda de diversos clientes. Nesses casos, podemos nos preocupar apenas com a quantidade de placas a serem cortadas, já que as sobras do corte podem ser recicladas. Já na indústria moveleira, as sobras de madeira são mais difíceis de ser recicladas, portanto tem-se a preocupação com o desperdício no corte.

Ambos PEB e PCB tem recipientes com as duas dimensões bem definidas. Existem problemas onde o tamanho do recipiente em uma das dimensões é ilimitado. Imagine um grande rolo de papel, como na Figura 2.4, que deve ser cortado em retângulos (itens) menores. Sua largura, em geral, é bem definida mas sua altura pode ser considerada infinita, pois a soma das alturas dos itens a serem cortados é muito menor que a altura da faixa. Esse problema é conhecido como Problema de Empacotamento Bidimensional em Faixas (*Two Dimensional Strip Packing Problem*). Sua definição é:

PROBLEMA DE EMPACOTAMENTO BIDIMENSIONAL EM FAIXAS - PEBF

Seja S uma faixa de largura L e altura infinita e uma lista de itens retangulares $I = (r_1, \dots, r_n)$, onde cada item $r_i = (l_i, a_i)$ é tal que $l_i \in (0, L]$, para $i = 1, \dots, n$, 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.

O PEBF é muito comum na indústria do papel, de polímeros e metalurgia. Dada essa grande importância, esse trabalho visou a resolução deste, de forma exata.

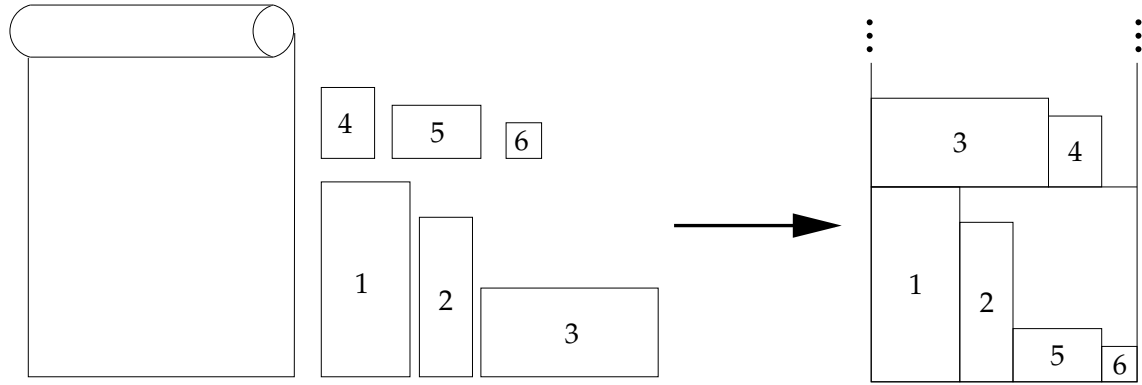


Figura 2.4: Exemplo de corte de um rolo de papel.

2.5 Problemas com mais de duas dimensões e variações no corte e empacotamento

Problemas de empacotamento tridimensionais são problemas muito comuns. Basta imaginar cargas de contêineres, caminhões ou armazéns. Problemas de corte são mais raros, mas podemos imaginar usinagem de peças, corte de madeira para fins específicos, corte de espumas para colchões, entre outros. O problema só é considerado tridimensional se três dimensões são relevantes ao processo de empacotamento.

Existem problemas que podem ser modelados com mais de três dimensões, por exemplo, as três dimensões físicas mais o tempo. Um exemplo de problema que pode ser modelado como um problema de empacotamento de mais de três dimensões é o problema de alocação de tarefas.

Fekete e Schepers [23, 24, 25] apresentam modelos genéricos para problemas de empacotamento de várias dimensões, utilizando uma caracterização baseada teoria dos grafos das soluções viáveis. Apresentam também os limitantes e um algoritmo exato associados a esta abordagem.

Além da dimensionalidade, os problemas de corte e empacotamento apresentam grande diversidade tanto quanto a espécie dos recipientes e itens, quanto ao tipo de

empacotamento propriamente dito. Na seção anterior, já definimos dois tipos de empacotamento: o empacotamento ortogonal, onde os lados dos itens são paralelos ou ortogonais aos lados dos recipientes, e o empacotamento não-ortogonal onde isso não é uma restrição. A Figura 2.3 mostra dois exemplos deste último.

O empacotamento também pode ser feito de itens com formatos diversos, como itens côncavos, itens com lados não regulares ou ainda itens cujas formas não são polígonos. Em outros problemas, é permitido que se mude a orientação do item, ou seja, que se rotacione este afim de otimizar o espaço do empacotamento. O Problema de Carregamento de Paletes/Estrados (*Pallet Loading Problem*) é um exemplo de um problema com estas características.

PROBLEMA DE CARREGAMENTO DE PALETES - PCP

Dados uma placa $R = (L, A)$ e valores x e y , empacotar o maior número de retângulos de tamanhos (x, y) ou (y, x) dentro de R de maneira ortogonal.

Como podemos observar, existe uma grande variedade de maneiras de se empacotar os retângulos na placa, já que cada um pode ser girado em 90° , com seus lados sempre ortogonais aos lados da placa. Acredita-se que este problema não pertence a classe NP. O PCP é um dos problemas que constam em aberto no site *The Open Problems Project* [16].

Outra consideração muito importante na indústria é de como os cortes são efetuados. Muitas vezes, o dispositivo de corte opera somente de forma longitudinal ou paralela aos lados do recipiente, e corta o material de uma ponta a outra. Este tipo de corte é conhecido como *corte guilhotinável*. A Figura 2.5 mostra o exemplo de três tipos de cortes. O tipo de corte (a) permite que a guilhotina seja utilizada trocando a orien-

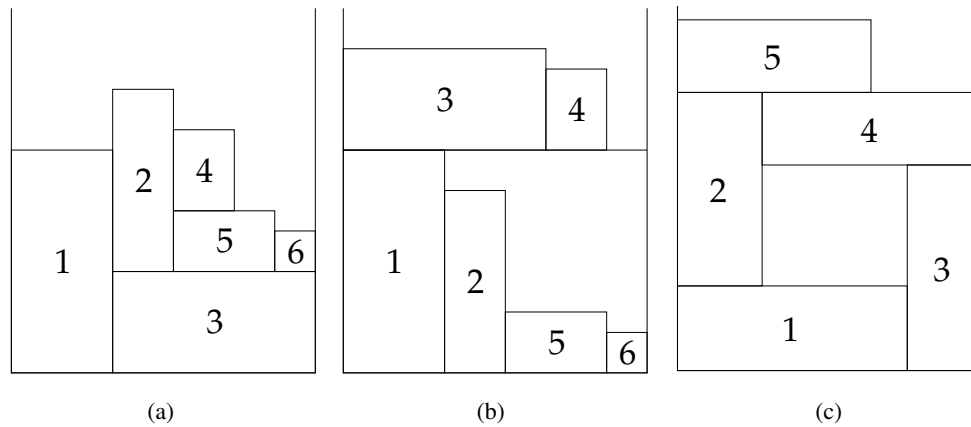


Figura 2.5: (a) Corte guilhotinável; (b) Corte guilhotinável com 2 estágios; (c) Corte não-guilhotinável.

tação da mesma em cada corte. Primeiro, o item 1 é cortado, em seguida 3, 2, 4 ou 6, e 5 finalmente, em um total de 4 ou 5 mudanças de direção da guilhotina. Se o custo de mudança de direção é muito alto, pode-se restringir este, resultando em um número máximo de estágios de corte. Um *estágio de corte* representa uma mudança na direção da guilhotina e uma sequência de cortes. Em (b), temos um estágio horizontal, que separa os itens 1, 2, 5 e 6 dos itens 3 e 4, e, em seguida, um estágio vertical que separa os 1, 2, 5 e 6 uns dos outros, e os itens 3 e 4 um do outro. Note que é necessário mais um estágio para o corte das sobras. Este estágio, em geral, não é contabilizado como estágio de corte. Em (c) é impossível utilizar uma guilhotina, a não ser para separar o item 5. A disposição dos itens 1, 2, 3 e 4 impedem que a guilhotina corte o material de uma ponta a outra.

Os cortes guilhotináveis em estágios são muito comuns na indústria devido o intenso uso de guilhotinas. Considerando o corte do rolo de papel, onde os itens tem demanda unitária, podemos chamá-lo de *Problema de Empacotamento Bidimensional em Faixas com 2 Estágios* PEBF2. A Figura 2.4 mostra um exemplo deste tipo de empacotamento. Consideraremos o número depois da sigla de um problema como o número de estágios requeridos para corte. Por exemplo, o PEB2 e o PEBF2 são versões do PEB e o PEBF, respectivamente, que requerem que os cortes sejam feitos em 2 estágios.

Outras restrições comuns são relativas ao sortimento dos itens e seu tipo. Imagine a carga de veículos com produtos alimentícios e produtos de limpeza. Tais produtos, em geral, não devem ser empacotados juntos por questões de contaminação. Outra situação é o empilhamento de itens. Certos materiais não podem ser sobrepostos a outros, portanto a ordem de empacotamento é importante.

Muitas dessas restrições influenciam diretamente nos algoritmos utilizados para resolução. O Capítulo 4 mostra as influências do corte guilhotinado para o PEBF na construção de algoritmos polinomiais aproximados, e o Capítulo 5 mostra a influência de restrições de empacotamento na geração de soluções heurísticas e na geração de colunas/padrões.

Capítulo 3

Técnicas de Resolução

3.1 Algoritmos Aproximados

Um algoritmo de aproximação ou algoritmo aproximado é um método polinomial para um determinado problema tal que, dada qualquer instância deste, devolve uma solução com garantia de proximidade da solução ótima.

Seja P um problema e \mathcal{I} o conjunto de todas instâncias possíveis de P , \mathcal{A} um algoritmo e denote por $\mathcal{A}(I)$ o valor de uma solução produzida pelo algoritmo \mathcal{A} para a instância $I \in \mathcal{I}$, e por $\text{OPT}(I)$ o valor de uma solução ótima da instância $I \in \mathcal{I}$. Dizemos que \mathcal{A} é um algoritmo que tem um fator de aproximação absoluto α ou é α -aproximado se

$$\mathcal{A}(I) \leq \alpha \cdot \text{OPT}(I), \quad \forall I \in \mathcal{I}$$

ou,

$$\mathcal{A}(I) \geq \alpha \cdot \text{OPT}(I), \quad \forall I \in \mathcal{I}$$

caso P seja de minimização ou maximização, respectivamente. Do mesmo modo, temos $\alpha \geq 1$ nos problemas de minimização e $\alpha \leq 1$ nos problemas de maximização. Se α não depende do tamanho da instância, dizemos que o algoritmo tem um fator de aproximação constante. Tomaremos por padrão problemas de minimização, já que as definições são análogas para ambos. Quando nos referirmos aos problemas de maximização, isto será feito explicitamente.

Dizemos que o algoritmo \mathcal{A} tem um fator de aproximação assintótico se

$$\mathcal{A}(I) \leq \alpha \cdot \text{OPT}(I) + C, \quad \forall I \in \mathcal{I}$$

para alguma constante C . A maioria dos algoritmos aproximados para problemas de corte e empacotamento apresentam fatores de aproximação assintóticos.

Dizemos que um algoritmo \mathcal{A} é um esquema de aproximação em tempo polinomial (PTAS, do inglês *Polynomial Time Approximation Scheme*) para um problema P se, para qualquer instância $I \in \mathcal{I}$ de P e cada $\varepsilon > 0$, temos

$$\mathcal{A}(I) \leq (1 + \varepsilon) \cdot \text{OPT}(I)$$

e o tempo de execução é limitado por um polinômio no tamanho de I . Se este tempo de execução é limitado por um polinômio no tamanho de I e $1/\varepsilon$, dizemos que o esquema de aproximação em tempo polinomial é total (FPTAS, do inglês *Fully Polynomial Time Approximation Scheme*).

Os algoritmos aproximados garantem a qualidade da solução e têm tempo de execução polinomial em relação à entrada. Considerando que a grande maioria dos problemas tratados são das classes NP-completo, NP-difícil ou classes superiores de complexidade, e que $P \neq NP$, o sacrifício na otimalidade da solução é justificado pelo tempo polinomial na execução do algoritmo.

Existem várias técnicas para a construção de algoritmos aproximados que se estendem desde técnicas puramente combinatoriais, passando por técnicas de programação linear (métodos primais-duais, relaxação lagrangeana e programação semi-definida), até métodos baseados em teoria dos jogos. Muitos problemas não podem ser aproximados dentro de um determinado fator por qualquer algoritmo polinomial. Estes resultados são conhecidos como resultados de inaproximabilidade. Uma excelente descrição destas técnicas e problemas a que se aplicam, assim como seus resultados de inaproximabilidade, podem ser encontradas em Vazirani [63].

3.2 Programação Dinâmica

A programação dinâmica é uma técnica de resolução de problemas através de seus subproblemas, utilizando uma sub-estrutura ótima para resolução. Diferentemente das técnicas baseadas em divisão e conquista, a programação dinâmica consiste em armazenar soluções de subproblemas e construir uma solução de problemas maiores a partir dos menores em uma estratégia *bottom-up*.

O primeiro passo na construção de um algoritmo de programação dinâmica é a determinação de uma sub-estrutura ótima. Entenda-se por *sub-estrutura ótima* o fato de uma solução ótima conter soluções ótimas de subproblemas associados, ou seja, uma solução ótima é construída de soluções parciais também ótimas. Esta sub-estrutura é definida recursivamente, como nos algoritmos de divisão e conquista, mas os problemas resolvidos em uma iteração compartilham as soluções dos subproblemas. Por isso na maioria das vezes, um algoritmo de programação dinâmica é executado sobre uma tabela de soluções parciais.

Vejam os casos do problema da mochila onde a capacidade da mochila e os pesos dos itens são inteiros (Seção 2.3). Consideremos que os itens estão numerados de $1, \dots, |I|$. Sejam $1 \leq i \leq |I|$, $0 \leq c \leq C$, e $f_i(c)$ o valor da solução ótima para o subproblema dado pelos itens $1, \dots, i$ e a mochila de capacidade c . Se considerarmos apenas o primeiro item, $f_1(c) = 0$ para $c = 0, \dots, p_1 - 1$, já que o item não cabe em nenhuma destas mochilas, e $f_1(c) = v_1$ para $c = p_1, \dots, C$, já que temos apenas um item. Então, para empacotar os outros itens, temos a recorrência

$$f_i(c) = \begin{cases} f_{i-1}(c) & \text{para } c < p_i, \\ \max(f_{i-1}(c), f_{i-1}(c - p_i) + v_i) & \text{caso contrário.} \end{cases}$$

A primeira equação traz o resultado do empacotamento das mochilas de tamanho no máximo $p_i - 1$, ou seja, não considera o empacotamento do item i . A segunda equação considera o item i e compara o valor de seu empacotamento com o valor do empacotamento sem considerar este. O maior valor é escolhido. Note que $f_{i-1}(c - p_i)$ é o valor do empacotamento da mochila de tamanho $c - p_i$ sem o item i , mas reservando espaço na mochila para este. Então, o valor é acrescido de v_i , preenchendo a mochila com o item i . Assim, a equação completa $f_{i-1}(c - p_i) + v_i$ considera o empacotamento do item i . Desta forma, podemos construir o Algoritmo 1 que foi originalmente exposto por Gilmore e Gomory [29] (veja Martello e Toth [46]).

Em geral, os algoritmos baseados em programação dinâmica necessitam de grande quantidade de memória por causa da tabulação dos dados. No Algoritmo 1, a situação se torna crítica se o tamanho da mochila é muito grande. Em verdade, este algoritmo é pseudo-polinomial de tempo $O(nC)$ onde n é o número de itens e C o tamanho da mochila.

Podemos obter um algoritmo polinomial a partir do Algoritmo 1, dentro de um certo fator de aproximação. Ibarra e Kim [34] obtiveram um esquema de aproximação totalmente polinomial (FPTAS) para o problema da mochila onde os valores dos itens são redefinidos, através de um parâmetro de erro, que limitado polinomialmente no tamanho da instância, permite construir um algoritmo de programação dinâmica polinomial. Para mais detalhes, vide Vazirani [63].

Para mais detalhes sobre programação dinâmica, veja Cormen *et al.* [11].

3.3 Programação Linear Inteira

A programação linear inteira é uma restrição da programação linear onde as variáveis pertencem a um domínio inteiro. A programação linear pode ser resolvida em tempo polinomial (Papadimitriou e Steiglitz [49]), mas isso nem sempre é verdade quando

Algoritmo 1: Programação dinâmica para o Problema da Mochila

Entrada: Um conjunto de itens I e a capacidade da mochila C .**Saída:** Uma matriz m contendo a solução.

```

1  Seja  $n$  o número de itens ( $n \leftarrow |I|$ );
2  Seja  $m$  uma matriz de dimensões  $(n + 1) \times (C + 1)$ ;
3  para  $c = 0$  até  $C$  faça
4  |    $m[0, c] \leftarrow 0$ ;
5  fim
6  para  $i = 1$  até  $n$  faça
7  |    $m[i, 0] \leftarrow 0$ ;
8  |   para  $c = 1$  até  $C$  faça
9  |   |   se  $p_i \leq c$  então
10 |   |   |   se  $m[i - 1][c] < v_i + m[i - 1, c - p_i]$  então
11 |   |   |   |    $m[i, c] \leftarrow v_i + m[i - 1, c - p_i]$ ;
12 |   |   |   senão
13 |   |   |   |    $m[i, c] \leftarrow m[i - 1, c]$ ;
14 |   |   fim
15 |   senão
16 |   |    $m[i, c] \leftarrow m[i - 1, c]$ ;
17 |   fim
18 fim
19 fim
20 devolva  $m$ .
```

tratamos de programas lineares inteiros. Portanto, são utilizadas relaxações onde as restrições de integralidade das variáveis são abandonadas. Em geral, essas relaxações apresentam resultados fracionários¹ que podem ser utilizados como limitantes duais.

Existem vários métodos para resolução de programação linear. Entre eles, os três mais conhecidos são: o método simplex, o método de pontos interiores e o método dos elipsóides. Destes, o mais utilizado é o método simplex. A razão de sua utilização é que, embora seja um algoritmo não polinomial (veja Klee e Minty [37]), o simplex é muito rápido na grande maioria dos casos, apresenta informações primais e duais do problema e a reotimização de um problema pré-otimizado com algumas alterações é simples. Chvátal [8] e Bazaraa *et al.* [2] apresentam o algoritmo simplex com mais detalhes. O método dos pontos interiores é um método polinomial, baseado em programação não-linear, cuja reotimização após uma modificação no problema não é simples na maioria das vezes. O método dos elipsóides foi o primeiro algoritmo po-

¹Se a matriz formada pelas restrições é totalmente unimodular, a solução ótima da relaxação é inteira, veja Wolsey [66]

linomial para resolução de programas lineares, mas apresenta complexidade elevada e difícil implementação. Este algoritmo trouxe importantes consequências na área de otimização combinatória (veja Schrijver [56]). Uma descrição deste algoritmo é dada por Papadimitriou e Steiglitz [49].

As relaxações lineares podem apresentar excelentes limitantes para soluções inteiras de problemas. É o caso do problema do corte unidimensional PCU (veja definições na Seção 2.3). Seja $n = |I|$. O vetor $P = (P_1, \dots, P_n) \in \mathbb{Z}_+^n$, representa um padrão de corte, ou seja, um conjunto de itens que podem ser empacotados em um recipiente. Cada P_i representa a quantidade de itens i no padrão P . Assim, uma formulação para o PCU é

$$\begin{aligned} & \text{minimize} && \sum_{P \in \mathcal{P}} \lambda_P \\ & \text{sob as restrições} && \sum_{P \in \mathcal{P}} P_i \lambda_P \geq d_i && \forall i \in I \\ & && \lambda_P \in \mathbb{Z}_+^n && \forall P \in \mathcal{P}, \end{aligned} \quad (\text{PLI})$$

onde λ_P é uma variável que indica quantas vezes o padrão P é utilizado e \mathcal{P} é o conjunto de todos padrões que respeitem $\sum_{i=1}^n p_i P_i \leq C$. Esta formulação é inteira, dada as restrições de integralidade da variável λ , e tem tamanho exponencial, já que o número de padrões possíveis é exponencial. Seja (PL) a relaxação de (PLI), tomando $\lambda_P \geq 0, \forall P \in \mathcal{P}$.

Muitas instâncias do PCU apresentam a propriedade IRUP (*Integer Round Up Property*), que diz que o valor de uma solução ótima inteira da Formulação (PLI) não é mais que o valor da relaxação linear (PL), arredondada para cima. Entretanto, Marcote [44] encontrou um contra-exemplo para esta conjectura, e Rietz *et al.* [50] mostraram famílias de instâncias onde essa propriedade falha. Scheithauer e Terno [54] propuseram uma alteração na propriedade IRUP, afirmando que o valor de uma solução ótima inteira não é mais que o valor da relaxação linear arredondada para cima adicionada de um. Esta nova conjectura é conhecida como *Modified Integer Round Up Property* (MIRUP).

Conjectura (MIRUP). *Sejam λ_{PL} e λ_{PLI} as soluções ótimas de (PL) e (PLI), e $\text{val}(\lambda_{PL})$ e $\text{val}(\lambda_{PLI})$, valores destas soluções, respectivamente. Então $\text{val}(\lambda_{PLI}) \leq \lceil \text{val}(\lambda_{PL}) \rceil + 1$.*

Muitos autores já procuraram por instâncias na tentativa de desprovar ou fortalecer esta conjectura segundo Scheithauer e Terno [55]. A maior diferença conhecida entre os valores de λ_{PL} e λ_{PLI} é de $1,1666\dots$, veja Rietz *et al.* [51].

Embora estes resultados pareçam promissores, algumas formulações apresentam resultados ruins como a Formulação (5.1) para o Problema de Empacotamento Bidimensional em Faixas com 2 estágios PEBF2, apresentada na Seção 5.1. Lodi *et al.*

[42] mostraram que a razão entre a solução ótima da relaxação desta formulação, e a solução ótima inteira, chamada de *gap de integralidade*, é de $1/2$. Portanto, melhores formulações devem ser feitas para que essa diferença seja menor, conseqüentemente melhorando o limitante.

3.4 Geração de Colunas

3.4.1 Introdução

Formulações com um número grande de variáveis são muito custosas para resolução: apenas a geração de todas variáveis possíveis já toma um tempo geralmente impraticável. Ainda assim, existem várias razões para se utilizar este tipo de formulação, segundo Barnhart *et al.* [1]:

- Uma formulação com um grande número de variáveis pode apresentar limitantes melhores que uma formulação compacta;
- Uma formulação com um grande número de variáveis pode eliminar simetrias que são prejudiciais ao desempenho do algoritmo de *branch-and-bound*;
- A decomposição do problema em um problema mestre e subproblemas pode apresentar interpretações naturais para o problema decomposto, incorporando informações importantes;
- Uma formulação com um grande número de variáveis pode ser a única escolha.

O algoritmo simplex trabalha com uma base do problema e, a cada iteração, procura por uma variável para entrar nesta, tal que melhore o valor da solução. Esta fase é conhecida como *pricing* (para mais detalhes, veja Chvátal [8]). Assim, o simplex deve analisar todas variáveis disponíveis e escolher aquela com melhor custo reduzido. Mas a simples enumeração de todas estas é praticamente impossível. Tanto o limite de espaço como de tempo são rapidamente atingidos em grandes formulações. Em geral, estas formulações têm número exponencial de variáveis. Assim, o sistema deve ser resolvido considerando um pequeno número destas, ou seja, a cada iteração do simplex, um pequeno número de variáveis deve estar em foco. Mas reside a questão de quais variáveis fornecer ao sistema de modo que ele atinja uma solução ótima. Muitas delas não fazem parte de uma solução ótima. Em verdade, o número é bem pequeno se comparado ao total de variáveis disponíveis.

Uma técnica utilizada para a prospecção de variáveis se chama *geração de colunas*, onde variáveis com custos reduzidos atrativos são geradas e fornecidas ao sistema.

Uma variável com custo reduzido atrativo é capaz de melhorar a qualidade da solução até então encontrada (mesmo que seja apenas o valor da relaxação). Se consideramos um problema de minimização, a variável gerada deverá ter custo reduzido negativo, capaz de diminuir o valor da solução. Em problemas de maximização, o custo reduzido deve ser positivo. O nome geração de colunas vem do fato que cada variável é uma coluna no tableau do método simplex utilizado para resolução de sistemas lineares. Assim, os termos “coluna” e “variável” podem ser intercambiáveis. Chvátal [8] mostra o funcionamento do algoritmo simplex e mostra a relação da fase de *pricing* e a geração de colunas.

A Figura 3.1 mostra um esquema de um algoritmo de geração de colunas para um problema de minimização. O sistema é iniciado com um conjunto de variáveis válidas tais que proporcionem a viabilidade do sistema, ou seja, pelo menos uma solução válida seja possível de ser encontrada, ou exista uma base viável. Então, o sistema é resolvido até a otimalidade com as colunas disponíveis. Após isso, é construído um subproblema com os custos duais do sistema que é responsável pela geração de colunas válidas. Ele deve devolver uma variável cujo custo reduzido seja negativo. Se isto acontece, a variável é inserida no sistema, que é reotimizado, recomeçando o ciclo (esta descrição é justamente a fase de *pricing* do simplex). Caso contrário, encontramos uma solução ótima da relaxação e esta é devolvida.

Muitas vezes, o problema que deve ser resolvido para a geração de colunas com custo reduzido atrativo, é NP-difícil. Uma maneira de resolvê-lo é gerar colunas de forma heurística ou aproximada. Entretanto, incorre o fato que estes algoritmos podem devolver colunas que não apresentam custo reduzido vantajoso. Como as colunas fornecidas até este ponto foram geradas de forma heurística ou aproximada, a relaxação pode não se encontrar em uma solução ótima fracionária, pois não temos garantia que a relaxação contém as melhores colunas. Invariavelmente, teremos que utilizar um algoritmo exato para gerar a melhor coluna possível neste ponto e, potencialmente, esse processo pode ser lento. Em vista disso, a geração de colunas é melhor aproveitada quando o problema a ser tratado apresenta algumas propriedades que tornam sua resolução rápida.

A geração de colunas muitas vezes é utilizada juntamente com um algoritmo de *branch-and-bound*, e recebem juntos o nome de algoritmo de *branch-and-price*. As ramificações feitas pelo algoritmo podem mudar profundamente o problema de geração de colunas, o que pode levar a sofisticados e complicados geradores de colunas. A Seção 3.6 mostra algumas considerações a respeito da ramificação e geração de colunas. Podemos observá-las também nas seções que tratam deste assunto no Capítulo 5.

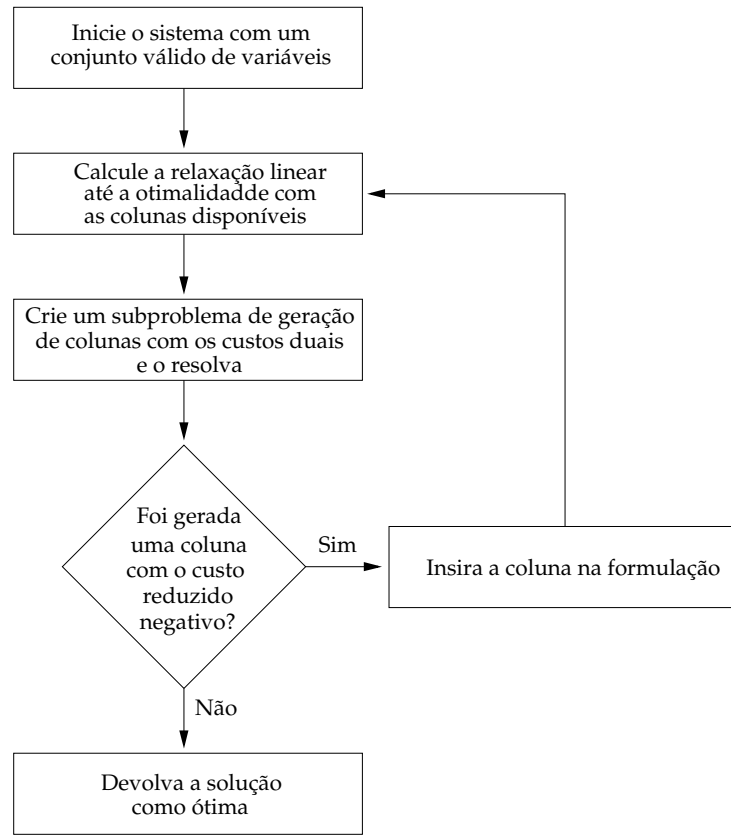


Figura 3.1: Esquema de geração de colunas para um problema de minimização.

3.4.2 A Decomposição de Dantzig-Wolfe

A geração de colunas está ligada a formulações com um grande número de variáveis. Muitas dessas formulações são mais restritivas que suas equivalentes de tamanho polinomial. Uma maneira de criar uma formulação mais restritiva a partir de uma mais relaxada é aplicar a decomposição de Dantzig-Wolfe.

Tomemos a seguinte definição:

Definição: Dado um conjunto $X \subseteq \mathbb{R}^n$, um ponto $x \in \mathbb{R}^n$ é uma *combinação convexa* de pontos de X se existe um conjunto finito de pontos $\{x^i\}_{i=1}^t$ em X e um $\lambda \in \mathbb{R}_+^t$ com $\sum_{i=1}^t \lambda_i = 1$ e $x = \sum_{i=1}^t \lambda_i x^i$. A *envoltória convexa* de X , denotada por $\text{conv}(X)$, é o conjunto de todos os pontos que são combinações convexas de pontos em X .

A decomposição de Dantzig-Wolfe foi proposta por Dantzig e Wolfe [12], e pode aplicar-se à programação inteira. Escolhemos um subconjunto de restrições como subsistema e consideramos implicitamente as soluções inteiras deste, através do poliedro X . Reformulamos o programa inteiro original trocando suas variáveis por combina-

ções convexas de um conjunto finito de pontos pertencentes a X e combinações lineares de um conjunto finito de raios pertencentes a X . Assim, obtemos uma nova formulação, chamada de *formulação mestre*, que é mais restritiva já que consideramos a envoltória convexa de X implicitamente. Considerar a envoltória convexa de X , significa resolver na otimalidade o problema descrito pelo poliedro X . Gilmore e Gomory [27] foram pioneiros na utilização desta técnica aplicando-a ao Problema de Empacotamento Unidimensional.

Quando decompomos uma formulação, conseguimos cortar vários pontos fracionários da formulação original, já que apenas soluções que respeitem a integralidade do poliedro X são válidas.

Exemplo 3.1: Consideremos o problema do empacotamento unidimensional PEU descrito na Seção 2.3. No pior caso, cada item deve ser empacotado em um recipiente. O número de recipientes pode ser limitado pelo número de itens. Consideremos n recipientes, onde $n = |I|$. Sejam as variáveis

$$y_j = \begin{cases} 1 & \text{se o recipiente } j \text{ faz parte da solução,} \\ 0 & \text{caso contrário} \end{cases} \quad \text{para } j = 1, \dots, n$$

e

$$x_{ij} = \begin{cases} 1 & \text{se o item } i \text{ está empacotado no recipiente } j, \\ 0 & \text{caso contrário} \end{cases} \quad \forall i \in I, j = 1, \dots, n.$$

Uma formulação do PEU é a seguinte:

$$\text{minimize} \quad \sum_{j=1}^n y_j \quad (3.1a)$$

$$\text{sob as restrições} \quad \sum_{j=1}^n x_{ij} = 1 \quad \forall i \in I \quad (3.1b)$$

$$\sum_{i \in I} p_i x_{ij} \leq C y_j \quad \text{para } j = 1, \dots, n \quad (3.1c)$$

$$y_j \in \{0, 1\} \quad \text{para } j = 1, \dots, n \quad (3.1d)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, j = 1, \dots, n \quad (3.1e)$$

onde a Função Objetivo (3.1a) minimiza a quantidade de recipientes utilizados, a Restrição (3.1b) garante que todos os itens estão empacotados em algum recipiente, a Restrição (3.1c) garante que o total dos pesos dos itens empacotados em um recipiente não

exceda a capacidade deste, e as Restrições (3.1d) e (3.1e) garantem a integralidade das variáveis.

A relaxação desta formulação abandona as restrições de integralidade, considerando as variáveis como contínuas. Desta maneira, obtemos um valor fracionário que pode ser distante do valor da solução inteira ótima. De fato, é fácil ver que o valor ótimo da relaxação desta formulação é $\sum_{i=1}^n p_i/C$. Isto implica que o *gap* de integralidade se aproxima de 1/2. Imagine que cada item i tem o peso $p_i = \lfloor C/2 + 1 \rfloor$. Considerando C tendendo ao infinito, a razão entre o valor da relaxação e o valor da solução ótima inteira tende a 1/2.

A Restrição (3.1c) é típica de problemas de empacotamento. Em verdade, é uma restrição do problema da mochila. Escolhemos esta para formar o poliedro

$$X = \left\{ x : \sum_{i \in I} p_i x_{ij} \leq C y_j, \quad \text{para } j = 1, \dots, n \right\}.$$

Note que o poliedro X é formado por vários conjuntos de restrições independentes entre si, cada um associado a um recipiente j . Assim

$$X = \bigcap_{j=1}^n X^j$$

onde

$$X^j = \left\{ x : \sum_{i \in I} p_i x_{ij} \leq C y_j \right\}.$$

Como todos recipientes têm tamanho igual, o conjunto de itens que pode ser aferido a um recipiente, pode ser aferido a qualquer outro recipiente. Seja $P = (P_1, \dots, P_n)$ o vetor binário que representa tal conjunto, onde P_i indica se o item i está ou não no conjunto. Este vetor é tido como *padrão de corte ou empacotamento*. Assim

$$\mathcal{P} = \left\{ P \in \{0, 1\}^n : \sum_{i \in I} p_i P_i \leq C \right\}. \quad (3.2)$$

Os vetores pertencentes a \mathcal{P} representam pontos inteiros contidos na envoltória convexa formada pelas restrições da mochila. Qualquer ponto P' desta envoltória pode ser escrito como uma combinação convexa dos pontos extremos de \mathcal{P} , ou seja,

$$P' = \sum_{P \in \mathcal{P}} P \lambda_P, \quad \text{onde} \quad \sum_{P \in \mathcal{P}} \lambda_P \leq 1.$$

Note que o padrão vazio pertence a \mathcal{P} (isto permite que a desigualdade seja menor ou igual e não apenas igual a 1).

Como a demanda dos itens no PEU é unitária, os conjuntos de itens que formam os padrões são disjuntos, e cada um destes não pode ser utilizado mais de uma vez. Portanto, o número de recipientes utilizados é justamente o número de padrões utilizados. Podemos reescrever a Função Objetivo (3.1a) como

$$\sum_{j=1}^n \sum_{P \in \mathcal{P}} \lambda_P^j$$

e a Restrição (3.1b) como:

$$\sum_{j=1}^n \sum_{P \in \mathcal{P}} P_i \lambda_P^j = 1, \quad \forall i \in I.$$

Então, temos a formulação mestre:

$$\begin{aligned} & \text{minimize} && \sum_{j=1}^n \sum_{P \in \mathcal{P}} \lambda_P^j \\ & \text{sob as restrições} && \sum_{j=1}^n \sum_{P \in \mathcal{P}} P_i \lambda_P^j = 1 && \forall i \in I && (3.3) \\ & && \sum_{P \in \mathcal{P}} \lambda_P^j \leq 1 \\ & && \lambda_P^j \geq 0. \end{aligned}$$

Esta formulação é mais restritiva que a Formulação (3.1) já que considera a envoltória convexa da mochila. ■

Segundo Vanderbeck [61], a motivação para se utilizar a decomposição de Dantzig-Wolfe é aproveitar a tratabilidade do subproblema (embora isso não seja trivial) para obter um limitante dual de qualidade, já que a formulação é mais restritiva.

Embora formulações resultantes da decomposição de Dantzig-Wolfe sejam mais restritivas, elas são muito grandes. Isso porque as variáveis do sistema são combinações de soluções válidas do subproblema resultante da decomposição. Como estes subproblemas, em sua grande maioria, são problemas combinatórios difíceis (NP-difíceis na maioria dos casos), o número de soluções pode ser exponencial, o que reflete em um número exponencial de variáveis na formulação mestre. Por isso, a técnica de geração de colunas é utilizada com frequência juntamente com formulações resultantes da decomposição.

3.4.3 Geração de colunas para problemas de corte e empacotamento

A maioria dos problemas de corte e empacotamento, senão todos, contêm restrições do problema da mochila envolvidas (não só unidimensional, como em d dimensões). Estas restrições são candidatas a formar subproblemas de onde é extraída a envoltória convexa (de fato, apenas alguns pontos desta envoltória). Se observamos atentamente, percebemos que as variáveis resultantes da decomposição são soluções do problema da mochila. Cada mochila resolvida, constitui um conjunto de itens a serem atribuídos a um recipiente. Chamamos essas soluções de *padrões de corte ou empacotamento*. Essa interpretação apresenta grandes vantagens semânticas que podem ser úteis na geração de colunas.

Exemplo 3.2: Tomemos o problema do empacotamento unidimensional PEU novamente. A Formulação (3.3) pode ser reescrita como a formulação mestre

$$\begin{aligned} & \text{minimize} && \sum_{P \in \mathcal{P}} \lambda_P \\ & \text{sob as restrições} && \sum_{P \in \mathcal{P}} P_i \lambda_P = 1 && \forall i \in I \\ & && \lambda_P \in \{0, 1\} && \forall P \in \mathcal{P} \end{aligned} \quad (3.4)$$

onde $P = (P_1, \dots, P_n)$ é um vetor binário representando um padrão, \mathcal{P} é o conjunto desses padrões e λ_P é a variável que indica se o padrão P é utilizado ou não. Esta simplificação pode ser feita já que não é necessário testar todas combinações para todos recipientes, pois o número destes é igual ao número de padrões a serem utilizados.

Como o tamanho de \mathcal{P} é potencialmente exponencial, testar todas colunas é impraticável. Temos que tomar os melhores padrões possíveis e fornecê-los à formulação mestre. O subproblema associado para gerar estes padrões é

$$\begin{aligned} & \text{maximize} && \sum_{i \in I} \pi_i P_i \\ & \text{sob as restrições} && \sum_{i \in I} p_i P_i \leq C \\ & && P_i \in \{0, 1\} && \forall i \in I \end{aligned} \quad (3.5)$$

onde π_i é o valor (dual) associado ao item i (note que $1 - \pi P$ é o custo reduzido da formulação mestre). A Formulação (3.5) remete a um problema da mochila, como podemos observar na Formulação (2.1) da Seção 2.3. ■

O processo de geração de colunas nos permite trabalhar com uma formulação mestre de tamanho reduzido, provendo variáveis quando necessárias. Essa formulação

mestre é conhecida como *formulação mestre restrita*. Como a prospecção de uma variável é dada pelos custos duais das restrições, utilizamos os custos duais da formulação mestre como valores dos itens no subproblema. Quando resolvido na otimalidade, o subproblema deve ser capaz de devolver um padrão cuja variável associada tenha custo reduzido atrativo, no caso do PEU, custo reduzido negativo. Se esse padrão não é encontrado, o sistema mestre se encontra em uma solução ótima. Note que esse processo é baseado na relaxação da formulação mestre. Desta maneira, a geração de colunas se coloca no lugar da fase de *pricing* do algoritmo simplex, e provê a solução ótima da relaxação sem a necessidade da enumeração de todas variáveis possíveis.

Gerar colunas para problemas de corte e empacotamento significa gerar padrões. Intuitivamente, os padrões são modelos ou guias que serão utilizados no processo industrial para gerar os itens no processo de corte, ou empacotar estes mesmos em algum recipiente. Mas por diversas razões, como o tipo de equipamento utilizado para empacotar um item ou cortar uma chapa, insumos cujo contato seja proibido, características do material utilizado no corte, etc, os padrões não podem ser gerados apenas baseados nas informações duais. Algumas destas considerações são mostradas na Seção 2.5.

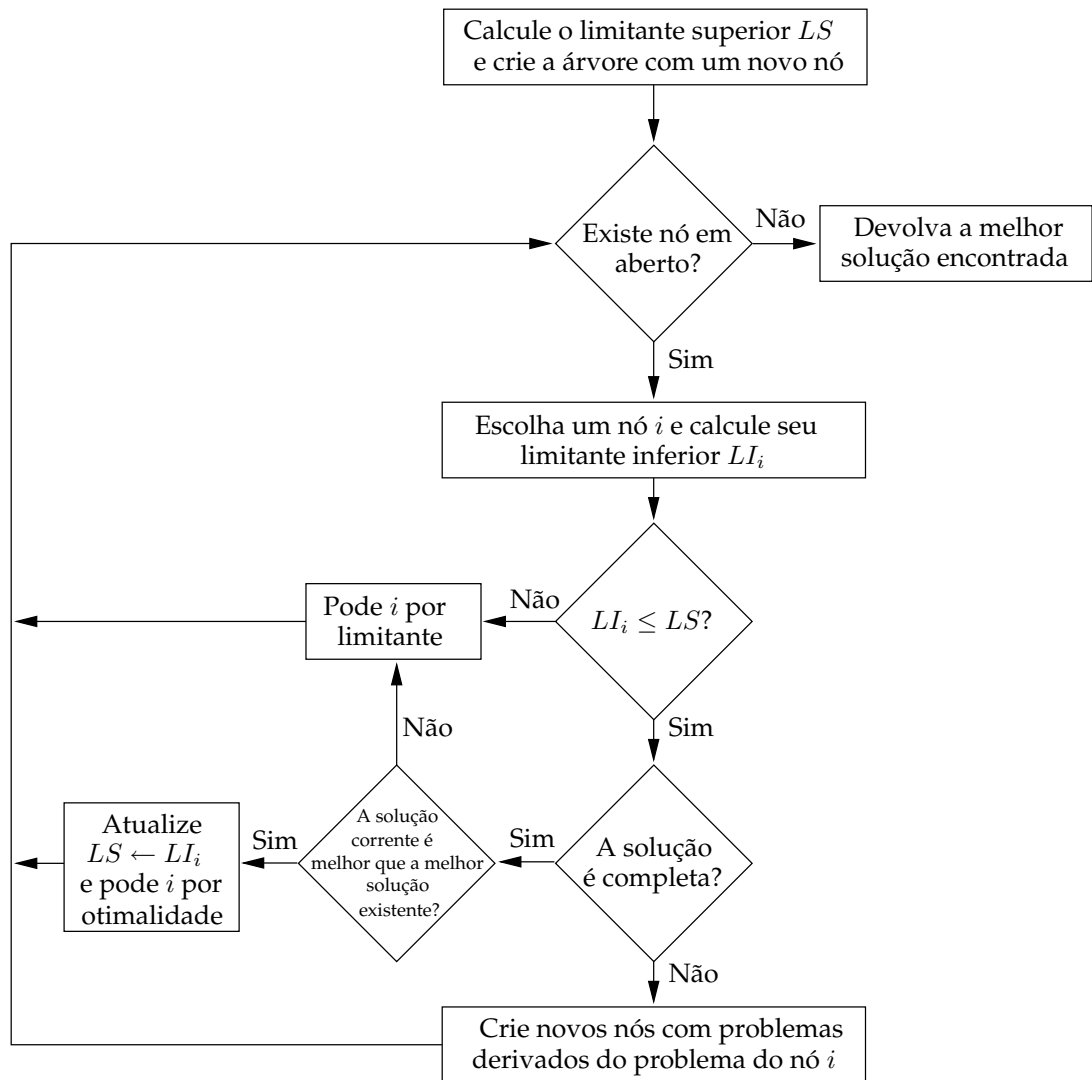
As restrições que ocorrem na prática podem dificultar muito o problema de geração de colunas. O problema da mochila (ou suas variantes nas diversas dimensões) é acrescido de várias restrições que podem impedir a utilização de estratégias mais rápidas como programação dinâmica ou algoritmos aproximados. Restrições deste tipo levam a condições de difícil tratamento que costumam a aparecer em problemas como busca de conjunto independente e coloração de grafos. Restrições de guilhotina podem impedir uma boa ocupação da área do padrão gerado. Portanto, os algoritmos de geração de colunas para problemas de corte e empacotamento tendem a ser complexos, dependendo do problema tratado.

3.5 O Método Branch-and-Bound

3.5.1 Introdução

Encontrar uma solução ótima para um problema NP-difícil pode ser extremamente custoso para grandes instâncias, dado o imenso número de combinações possíveis, o que torna a simples enumeração inviável. Existem combinações ou padrões de combinações que são ruins e sua inspeção pode ser dispensada. Partindo deste princípio, podemos construir um algoritmo que detecta esses padrões e ignora a inspeção de variações destes, que também são ruins.

Os algoritmos de *Branch-and-Bound* são algoritmos de enumeração que utilizam o princípio de exclusão de soluções ruins e suas variações. Por este fato, os algoritmos de

Figura 3.2: Esquema do método de *Branch-and-Bound* para problemas de minimização.

branch-and-bound também são conhecidos como algoritmos de enumeração implícita. A Figura 3.2 mostra um esquema de um algoritmo deste tipo para um problema de minimização.

O primeiro passo é delimitar a região de possíveis soluções. Para isso, o algoritmo calcula dois limitantes chamados de limitantes primais e duais, que dizem os valores-limite que uma solução pode atingir, ou seja, os valores máximo e mínimo que a instância do problema comporta. O objetivo é recalcular estes limitantes até que sejam iguais e possamos afirmar que a solução é ótima, ou sua diferença seja a mínima possível, o que limita o espaço de busca do algoritmo. Um limitante primal é um valor dado por uma solução válida para o problema, em geral, pior que o valor de uma solução ótima. Um limitante dual é um valor otimista que uma solução ótima pode atingir. Nos problemas de minimização, o limitante primal é considerado como limitante superior e o limitante dual como limitante inferior. Nos problemas de maximização, ocorre o inverso. Para mais detalhes, vide Wolsey [66].

Geralmente, para controlar o processo de enumeração da busca de soluções, o algoritmo constrói uma árvore, chamada de *árvore de ramificação* (ou de *branching*), onde cada nó representa uma partição no espaço de soluções do problema. Cada nó apresenta um conjunto de possíveis soluções (às vezes, nenhuma solução), sujeitas a certas restrições provenientes da ramificação, para as quais serão calculados novos limitantes. Este cálculo é conhecido como processo de *bound*.

Se em um determinado nó, temos uma solução completa válida tal que seu valor é igual ao limitante dual, então encontramos a solução ótima. Caso contrário, continuamos com as modificações no problema, construindo outras soluções parciais a partir da atual, que serão abrigadas em nós filhos do nó corrente. Este processo é conhecido como *ramificação* (*branching*).

Suponhamos que uma destas soluções parciais, embora válida, não é capaz de gerar soluções melhores que o limitante primal. Pela estrutura do problema, sabemos que qualquer solução gerada a partir desta não vai apresentar valor melhor que o limitante primal. Portanto, não há necessidade de avaliar a sub-árvore desta solução e podemos eliminar esse passo. Este processo é conhecido como *poda*. O algoritmo procura diminuir o espaço de busca, podando a árvore de soluções e limitando a sua geração.

Um dos fatores cruciais para o desempenho de um algoritmo de *branch-and-bound* é o cálculo dos limitantes. Um bom limitante primal é capaz de podar prematuramente ramos não-promissores, diminuindo o tamanho da árvore de busca. Estes limitantes podem ter custo elevado de cálculo e podem não cortar ramo nenhum, segundo Brassard e Bratley [6]. Mas, mesmo assim, seu cálculo faz-se necessário.

Em geral, são utilizadas heurísticas ou algoritmos aproximados para o cálculo de limitantes primais, na tentativa de se obter uma boa solução inicial. Uma das manei-

ras de calcular o limitante dual é resolver a relaxação do programa linear relativo ao problema no nó da árvore em questão, utilizando variáveis contínuas, processo que é detalhado na próxima seção.

Podemos perceber que a informação necessária (limitantes e restrições que particionam o espaço de soluções) para o funcionamento de um algoritmo de *branch-and-bound* está nas folhas da árvore. Por isso, a implementação destes algoritmos, na maioria das vezes, só considera as folhas construindo a árvore implicitamente.

3.5.2 O algoritmo de Branch-and-Bound e a Programação Linear

Uma boa forma de calcular o limitante dual de um problema é resolver a relaxação da formulação linear inteira deste, onde as restrições de integralidade das variáveis não são respeitadas. Desta maneira, a solução da relaxação, na maioria das vezes fracionária, pode ser usada como uma valiosa informação na hora da poda da árvore. A grande vantagem da utilização da programação linear é que ela pode ser resolvida em tempo polinomial no tamanho do programa linear. Em cada nó, o problema é reotimizado levando em conta suas modificações. Em geral, estas modificações são acréscimos de restrições ou variáveis ao problema. O primeiro trabalho que apresenta um algoritmo de *branch-and-bound* utilizando programação linear inteira é de Land e Doig [38], e os primeiros resultados de sucesso foram conseguidos por Little *et al.* [39] para o Problema do Caixeiro Viajante.

O algoritmo de *branch-and-bound* com programação linear utiliza a relaxação da formulação do problema em questão para calcular os limitantes duais. Em geral, quando o valor resultante das variáveis é fracionário, a ramificação é feita tomando uma variável, digamos y , com valor fracionário $y = v$, e considerando duas opções, uma com a limitação $y \leq \lfloor v \rfloor$ e outra com a limitação $y \geq \lceil v \rceil$. Note que, ajustar os limites das variáveis corresponde exatamente em construir uma solução parcial: cada ajuste de integralidade força a escolha de valores para uma solução válida. Para mais detalhes, veja Papadimitriou e Steiglitz [49], e Nemhauser e Wolsey [47].

3.6 O Método Branch-and-Price

O algoritmo de *branch-and-bound* com relaxações de programação linear que permite a geração de colunas e sua inserção na formulação é conhecido como *Branch-and-Price*.

A Figura 3.3 mostra um esquema para um algoritmo de *branch-and-price* para problemas de minimização. Como não temos todas as colunas disponíveis, devemos iniciar o sistema com uma solução válida, que nos fornecerá um conjunto restrito de

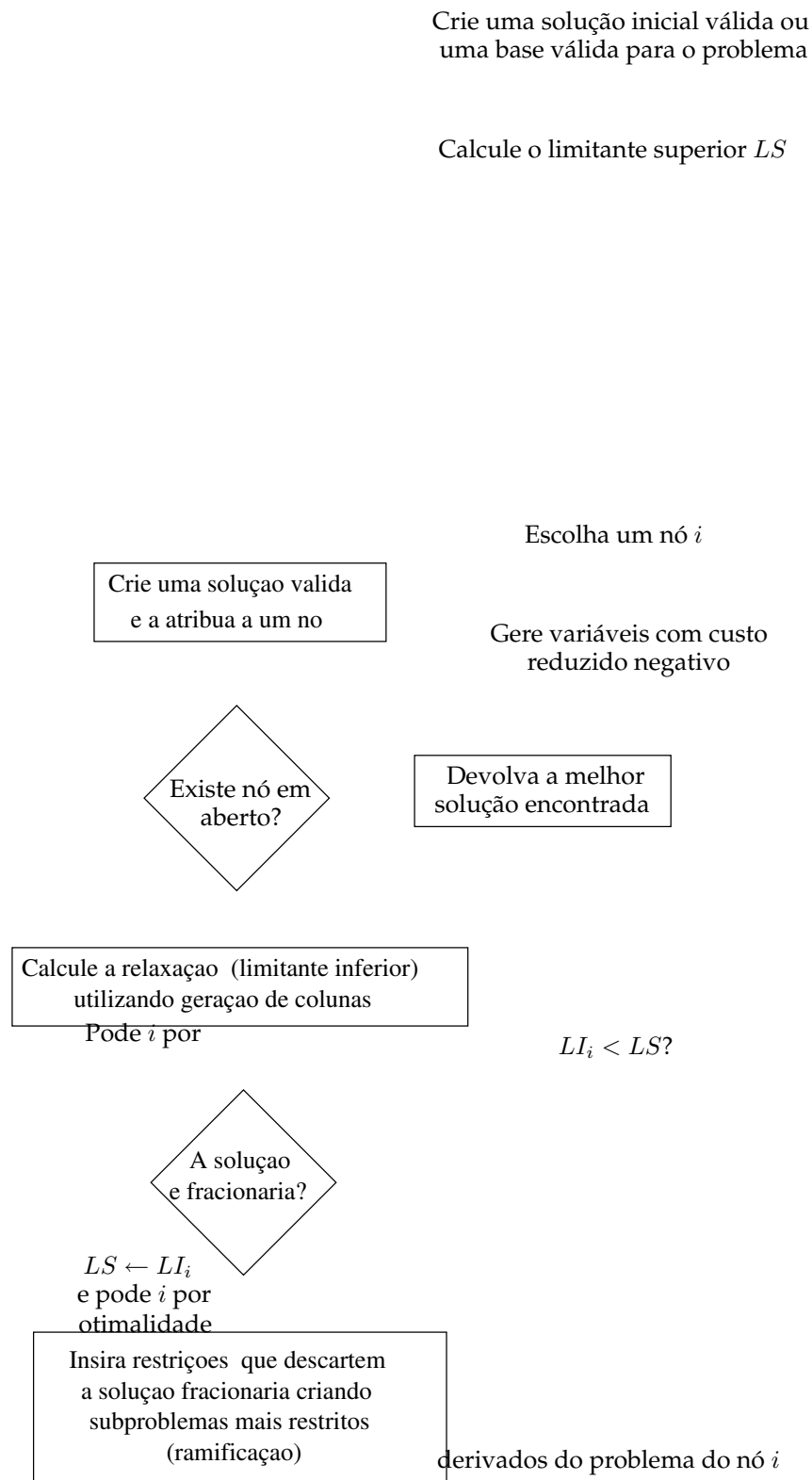


Figura 3.3: Esquema do método de *Branch-and-Price* para problemas de minimização.

variáveis, ou uma base válida. Como no algoritmo de *branch-and-bound* padrão, os limitantes são calculados e testados.

O teste sobre a solução atual deve ser baseada na otimalidade da relaxação. Como não temos todas as variáveis disponíveis, o valor da relaxação pode não ser o ótimo. Assim, temos que gerar as colunas que melhorem a qualidade da solução, de modo que leve a relaxação à otimalidade. O retângulo tracejado da Figura 3.3 destaca a fase de geração de colunas, que é descrita em detalhes na Seção 3.4. O restante do algoritmo procede da mesma maneira que um algoritmo de *branch-and-bound* padrão.

Os algoritmos de *branch-and-price* têm sido muito utilizados em diversos problemas. Vance aplicou esta técnica para o problema de empacotamento unidimensional [57] e para o problema de corte unidimensional [58], Desrosiers *et al.* [17] para problemas de roteamento, Vance *et al.* [59] para problemas de escalonamento de tripulação, Savelsbergh [53] para o problema de atribuição generalizada, Jeong *et al.* [35] para o problema da árvore de Steiner, entre outros. Barnhart *et al.* [1] a apresentam uma discussão sobre reformulação com grande números de variáveis e algoritmos de *branch-and-price*.

3.6.1 Considerações sobre algoritmos de Branch-and-Price

Ramificação

Embora seja promissor, o método *branch-and-price* esbarra em uma dificuldade crucial: o problema é modificado quando as restrições da ramificação são adicionadas no problema principal, o que desfigura os subproblemas (de geração de colunas), ou seja, seu tipo pode ser diferente do tipo da última iteração, implicando na destruição da estrutura que é explorada ou necessária para resolução eficiente destes subproblemas. Isto significa que o algoritmo utilizado para geração das colunas tende a ser mais complexo, tratando de vários casos, o que potencialmente aumenta o custo computacional. Portanto, é necessário que as regras de ramificação sejam elaboradas cuidadosamente para uma resolução eficiente dos subproblemas em cada nó da árvore de busca. Vanderbeck [61] e Vanderbeck e Wolsey [62] discutem algumas maneiras de ramificação para minimizar o impacto sobre a geração de colunas.

Geração de Colunas

O processo de gerar uma coluna pode ser trabalhoso. Em geral, utiliza-se um método rápido como um algoritmo aproximado ou uma heurística para que seja gerado uma coluna viável. A manipulação dessas colunas também pode ser trabalhosa, já que o processo de geração pode devolver uma coluna que já foi eliminada nas ramificações anteriores. Portanto, é necessário que haja um gerenciamento de colunas eficiente.

Outra dificuldade muito conhecida é o efeito de *tailing-off*, onde um grande número de iterações é necessário para provar a otimalidade. Potencialmente, isto pode ocorrer em cada nó da árvore onde o subproblema a ser resolvido é tipicamente um problema inteiro difícil (veja Vanderbeck e Wolsey [62]).

Capítulo 4

Algoritmos Aproximados e Heurísticas para o PEBF2

4.1 O algoritmo Next-Fit Decreasing Height

Existem muitos algoritmos aproximados que tratam de problemas de corte e empacotamento. Alguns destes algoritmos, para problemas com duas dimensões ou mais, são ditos algoritmos de níveis, uma vez que os itens são empacotados em níveis, da esquerda para direita, e o início de um novo nível coincide com o topo do item mais alto do nível anterior. A grande vantagem é que estes algoritmos são rápidos e geram padrões guilhotináveis.

O *Next-Fit Decreasing Height* (NFDH) é um dos mais tradicionais algoritmos aproximados para problemas de empacotamento bidimensionais, sendo derivado do *Next-fit* para casos unidimensionais (vide Lodi *et al.* [42]). O NFDH é um algoritmo de níveis *off-line*, já que precisa de conhecimento prévio das dimensões de todos os itens a serem empacotados. Seu princípio é simples e está descrito no Algoritmo 2. Primeiro, ordena-se os itens em ordem não-crescente de altura e cria-se um nível, empacotando os itens na ordem até que um destes não caiba no nível. Quando um item não couber em um nível, será criado um novo e o item empacotado aí. O empacotamento prossegue a partir desse novo nível.

Note que o NFDH gera padrões guilhotinados. Além disso, ele é um algoritmo muito simples e tem tempo linear de execução, a não ser pelo limitante da ordenação, que o torna $O(n \log n)$.

A Figura 4.1 mostra um exemplo de empacotamento onde foi aplicado o NFDH. Os itens numerados de acordo com sua altura, do mais alto para o mais baixo, e foram empacotados na sequência de suas alturas. Os itens 1 e 2 são empacotados no primeiro nível. Como o item 3 não cabe neste nível, é criado um novo e o item 3 empacotado

Algoritmo 2: *Next-Fit Decreasing Height***Entrada:** Um conjunto de itens I .**Saída:** Um conjunto de níveis.

```

1 Ordene os itens de  $I$  em ordem não-crescente de altura;
2 Crie um novo nível vazio e defina-o como nível corrente;
3 para cada  $i \in I$  faça
4   se  $i$  cabe no nível corrente (em largura) então
5     Coloque  $i$  no nível alinhado a esquerda;
6   senão
7     Crie um novo nível vazio e tome ele como corrente;
8     Coloque  $i$  no novo nível alinhado a esquerda;
9   fim
10 fim
11 devolva os níveis criados.

```

neste. Segue-se o empacotamento do item 4. O item 5 não cabe neste nível, portanto, é criado um novo nível para abrigá-lo. E o último a ser empacotado é o item 6. Embora, no primeiro nível coubessem juntos os itens 4 e 6, ou ainda, os itens 5 e 6, o algoritmo não os empacota aí pois o item 3 é o próximo a ser processado, forçando a criação de um novo nível por não caber no nível inferior.

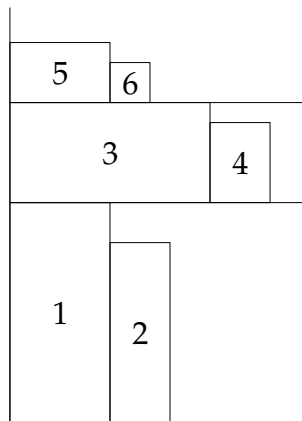


Figura 4.1: Empacotamento utilizando NFDH.

Essa característica seqüencial do NFDH, leva-o a uma perda significativa de área útil, que poderia ser utilizada para empacotar outros itens. Coffman *et al.* [9] provaram que

$$\text{NFDH}(I) \leq 2 \cdot \text{OPT}(I) + H,$$

onde H é a altura do item mais alto do empacotamento.

4.2 O algoritmo First-Fit Decreasing Height

O *First-Fit Decreasing Height* (FFDH) é um algoritmo similar ao NFDH. A diferença reside na escolha de onde empacotar um item. O item é empacotado no primeiro nível que couber, na ordem de geração dos níveis, alinhado a esquerda. Caso não caiba em nenhum nível, um novo nível é criado e o item empacotado aí. Como exige ordenação dos itens *a priori*, também é considerado um algoritmo *off-line*. O Algoritmo 3 mostra seu princípio.

Algoritmo 3: *First-Fit Decreasing Height*

Entrada: Um conjunto de itens I .

Saída: Um conjunto de níveis.

```

1 Ordene os itens de  $I$  em ordem não-crescente de altura;
2 Crie um novo nível vazio. Seja  $\mathcal{N}$  o conjunto de níveis;
3 para cada  $i \in I$  faça
4   | Seja  $N \in \mathcal{N}$ , na ordem em que foi criado, tal que  $i$  caiba em  $N$ ;
5   | se encontrou  $N$  então
6   |   | Coloque  $i$  em  $N$  alinhado a esquerda;
7   | senão
8   |   | Crie um novo nível vazio;
9   |   | Coloque  $i$  no novo nível alinhado a esquerda;
10  | fim
11 fim
12 devolva  $\mathcal{N}$ .
```

A Figura 4.2 mostra o empacotamento dos itens da Figura 4.1 utilizando o algoritmo FFDH. Os itens 1 e 2 são empacotados no primeiro nível e o item 3 no segundo nível, já que não cabe no primeiro. Ao tentar empacotar o item 4, o algoritmo detecta que o mesmo cabe no primeiro nível e então empacota ele ali. O item 5 já não cabe no primeiro nível, então, é empacotado no segundo, onde encaixa perfeitamente. O item 6 é empacotado no primeiro nível também, já que cabe no mesmo.

O algoritmo FFDH pode ser implementado em tempo $O(n \log n)$. Coffman *et al.* [9] provaram que a seguinte desigualdade é válida:

$$\text{FFDH}(I) \leq \frac{17}{10} \cdot \text{OPT}(I) + H,$$

onde H é a altura do item mais alto.

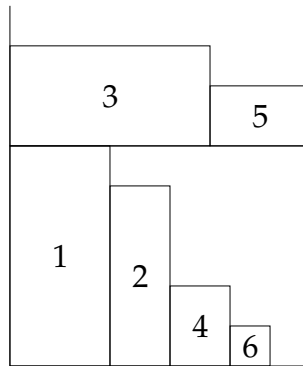


Figura 4.2: Empacotamento utilizando FFDH.

4.3 O algoritmo Best-Fit Decreasing Height

Embora muito utilizados, tanto o NFDH como o FFDH aparentam não utilizar eficientemente o espaço dos níveis. Uma alternativa é empacotar o item onde melhor couber. Este é o princípio que o *Best-Fit Decreasing Height* (BFDH) utiliza. Quando um item vai ser empacotado, o BFDH procura por um nível cujo espaço restante após o empacotamento do item é o menor possível. Consideramos aqui, não a área ocupada, mas a largura do item e largura restante do nível. Assim como os algoritmos anteriores, exige-se ordenação prévia dos itens. O Algoritmo 4 elucida o BFDH.

Algoritmo 4: *Best-Fit Decreasing Height*

Entrada: Um conjunto de itens I .

Saída: Um conjunto de níveis.

- 1 Ordene os itens de I em ordem não-crescente de altura;
 - 2 Seja \mathcal{N} o conjunto de níveis. Denote por S_N o espaço restante para empacotamento (em largura) do nível N ;
 - 3 **para cada** $i \in I$ **faça**
 - 4 Seja $N \in \mathcal{N}$ tal que $S_N - largura(i) \geq 0$ é mínimo;
 - 5 **se encontrou** N **então**
 - 6 Coloque i em N alinhado a esquerda;
 - 7 **senão**
 - 8 Crie um novo nível;
 - 9 Coloque i neste nível alinhado a esquerda;
 - 10 **fim**
 - 11 **fim**
 - 12 **devolva** \mathcal{N} .
-

Tomemos os itens utilizados nos exemplos anteriores: utilizando o BFDH, o empacotamento será o mostrado na Figura 4.3. Os itens 1, 2 e 3 serão empacotados como no NFDH e FFDH. O item 4 será empacotado no segundo nível, pois o espaço restante será menor que no primeiro nível, uma vez que, o item 3 é mais largo que os itens 1 e 2 juntos. O item 5 será então empacotado no primeiro nível pois não caberá no segundo. O item 6 cabe em ambos níveis, mas será empacotado no primeiro, que caberá perfeitamente.

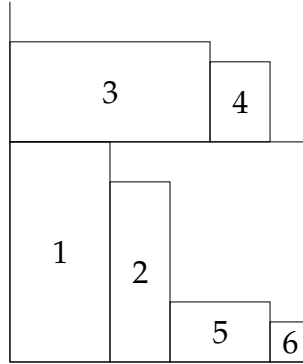


Figura 4.3: Empacotamento utilizando BFDH.

O BFDH pode ser implementado em tempo $O(n \log n)$. Embora mais sofisticado, não apresentou ganhos significativos nas instâncias testadas. Os resultados podem ser comparados na Seção 6.4.

4.4 O algoritmo da Mochila Gulosa

Embora rápidos e relativamente eficazes, o NFDH, o FFDH e o BFDH não consideram a área dos itens. Um algoritmo que considera este fato pode ser promissor, já que tentaria colocar itens que maximizassem a utilização da área de um nível. Lodi *et al.* [41] propuseram 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. Sejam a_i e l_i a altura e largura do item i , respectivamente, i' o item mais alto e L a largura da faixa. A mochila terá o tamanho $L - l_{i'}$ e os valores dos itens serão dados por $v_i = a_i \cdot l_i$. O Algoritmo 5 retrata essa proposta.

Este algoritmo usa uma estratégia gulosa, gerando a cada iteração um empacotamento de um nível com a maior ocupação de área com os itens não empacotados.

O algoritmo da mochila gulosa apresenta como subproblemas, instâncias do problema da mochila, já provado ser NP-difícil [26] e, portanto, não pode ser resolvido em tempo polinomial, a não ser que $P = NP$. Lodi *et al.* [41] propõem a utilização

Algoritmo 5: Mochila Gulosa

Entrada: Um conjunto de itens I e largura da faixa L .
Saída: Um conjunto de níveis.

```

1 Ordene os itens de  $I$  em ordem não-crescente de altura;
  // Calcula a área de cada item
2 para cada  $i \in I$  faça
3    $v_i \leftarrow a_i \cdot l_i$ ;
4 fim
5 enquanto  $I \neq \emptyset$  faça
6   Seja  $i \in I$  o item mais alto;
7   Crie um novo nível e empacote o item  $i$  neste;
8    $I \leftarrow I \setminus \{i\}$ ;
9   Seja  $\mathcal{M}$  uma instância do problema da mochila de tamanho  $C = L - l_i$  e itens
    dados pelo conjunto  $I$ ;
10  Resolva  $\mathcal{M}$ ;
11  Seja  $J$  o conjunto de itens devolvido pela resolução de  $\mathcal{M}$ ;
12  Coloque os itens de  $J$  no nível;
13   $I \leftarrow I \setminus J$ ;
14 fim
15 devolva os níveis gerados.

```

de algumas iterações de um algoritmo não-polinomial para resolução aproximada do problema da mochila (alguns destes algoritmos podem ser encontrados em Martello e Toth [46]). Nós utilizamos o Algoritmo 1 de programação dinâmica mostrado na Seção 3.2.

O algoritmo da mochila gulosa apresentou resultados muito parecidos com o FFDH e BFDH (vide Seção 6.4). Desta maneira, modificamos o cálculo dos valores dos itens, não apenas considerando sua área, mas também uma pequena depreciação neste, 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 aplicada uma função de amortização do valor (área) do item. Desta maneira, buscamos priorizar itens mais altos em detrimento aos mais largos.

Seja $B \in (0, 1)$ o limiar e $PT_i \in (0, 1]$ 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 $v(i) = area(i) \cdot e^{10 \cdot (PT_i - B)}$. Os resultados podem ser comparados na Seção 6.4

4.5 Outras abordagens

Hopper e Turton [31] apresentam uma revisão da aplicação de várias metaheurísticas aos problemas de empacotamento bidimensional em faixas, destacando diversas variações utilizando algoritmos genéticos. Os autores também comentam sobre soluções que empregam algoritmos baseados em *simulated annealing*, busca tabu e redes neurais. Embora este trabalho não trate especificamente do PEBF2, é um bom resumo de problemas e soluções similares ao PEBF2. Lodi *et al.* [41] apresentam algumas heurísticas para problemas de empacotamento bidimensional e um algoritmo de busca tabu para o problema de empacotamento bidimensional.

Capítulo 5

Método Branch-and-Price para o PEBF2

5.1 Formulações

A primeira formulação para problemas de corte e empacotamento é devida a Kantorovich [36]. Posteriormente, muitos trabalhos apresentaram diferentes formulações às mais diversas variantes destes problemas. Dentre elas, destaca-se a formulação do trabalho pioneiro de Gilmore e Gomory, utilizando uma formulação para o problema de empacotamento unidimensional PEU [27] e para o problema de empacotamento bidimensional em 2 estágios PEB2 [28], ambas de tamanho exponencial no número de variáveis.

Lodi *et al.* [43] apresentam formulações para compactas o PEB2 e para o problema do empacotamento bidimensional em faixas com 2 estágios PEBF2. Reproduzimos aqui a formulação para o PEBF2. Para conseguir esta formulação, fazemos as seguintes considerações:

Observação: Para qualquer solução ótima do PEBF2, existe uma solução equivalente tal que:

- (i) O primeiro item (mais a esquerda) empacotado em cada nível é o item mais alto do nível;
- (ii) O primeiro nível (mais baixo) empacotado na faixa é o nível mais alto nesta.

Esta formulação contém dois conjuntos de variáveis. Sem perda de generalidade, assumimos que $a_1 \geq a_2 \geq \dots \geq a_n$, onde a_i é a altura do item i pertencente ao conjunto de itens I . Assumimos $n = |I|$ possíveis níveis, cada um associado a um item i que inicializa este, tendo, portanto, altura correspondente a_i . O primeiro conjunto de

variáveis indica se um item inicia ou não um nível:

$$y_j = \begin{cases} 1 & \text{se o item } j \text{ inicializa o nível } j, \\ 0 & \text{caso contrário} \end{cases} \quad \text{para } j = 1, \dots, n$$

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

$$x_{ij} = \begin{cases} 1 & \text{se o item } i \text{ está empacotado no nível } j, \\ 0 & \text{caso contrário} \end{cases} \quad \text{para } i = 1, \dots, n \text{ e } i > j.$$

A formulação é apresentada a seguir:

$$\text{minimize} \quad \sum_{j=1}^n a_j y_j \quad (5.1a)$$

$$\text{sob as restrições} \quad \sum_{j=1}^{i-1} x_{ij} + y_i = 1 \quad \text{para } i = 1, \dots, n \quad (5.1b)$$

$$\sum_{i=j+1}^n l_i x_{ij} \leq (L - l_j) y_j \quad \text{para } j = 1, \dots, n-1 \quad (5.1c)$$

$$y_j \in \{0, 1\} \quad \text{para } j = 1, \dots, n \quad (5.1d)$$

$$x_{ij} \in \{0, 1\} \quad \text{para } j = 1, \dots, n-1 \text{ e } i > j. \quad (5.1e)$$

A Função Objetivo (5.1a) minimiza a altura do empacotamento, a Restrição (5.1b) força que um item seja empacotado apenas uma vez, ou iniciando um nível ou como parte deste nível, e a Restrição (5.1c) limita a ocupação de itens em um nível por sua largura. As Restrições (5.1d) e (5.1e) garantem a integralidade das variáveis.

Esta formulação esbarra em uma dificuldade: o *gap* entre a solução ótima da relaxação e a solução ótima inteira é pelo menos $1/2$ (Lodi *et al.* [43]) e, portanto, não oferece um limitante de boa qualidade. Uma maneira de “apertar” a formulação é aplicar a decomposição de Dantzig-Wolfe.

O conjunto de padrões que podem ser aferidos a um nível é igual, já que não há distinção entre os níveis, embora cada nível tenha um item associado para iniciá-lo. Assim, obtemos a formulação mestre para o PEBF2:

$$\begin{aligned} & \text{minimize} \quad \sum_{P \in \mathcal{P}} a_P \lambda_P \\ & \text{sob as restrições} \quad \sum_{P \in \mathcal{P}} P_i \lambda_P = 1 \quad \text{para } i = 1, \dots, n \quad (5.2) \\ & \quad \quad \quad \lambda_P \in \{0, 1\} \quad \forall P \in \mathcal{P} \end{aligned}$$

onde $P = (P_1, \dots, P_n)$ é um vetor binário representando a configuração de um nível, ou seja, um padrão. O conjunto \mathcal{P} contém todos os padrões P que satisfazem $\sum_{i \in I} l_i P_i \leq L$ e definimos $a_P = \max(P_i \cdot a_i)_{i \in \{1, \dots, n\}}$, ou seja, a altura do item mais alto em P . A variável λ_P indica se o padrão é utilizado ou não na solução, e P_i se o item i está empacotado no padrão P . Como as demandas dos itens são unitárias, nenhum padrão se repetirá no empacotamento e cada nível terá uma configuração própria. Podemos então, assumir que os termos padrão e nível se referem a uma configuração de itens.

A Formulação (5.2) é mais restritiva que a Formulação (5.1), uma vez que as soluções fracionárias que não são combinações convexas de soluções inteiras de problemas da mochila não são viáveis. Vance [58] aplicou a decomposição de Dantzig-Wolfe para o problema de corte e estoque unidimensional PCU, obtendo formulações similares às apresentadas aqui.

Embora mais restritiva, esbarramos numa formulação de tamanho exponencial devido ao imenso número de padrões possíveis. De fato, o número de padrões é próximo do número de combinações entre os itens. Assim, devemos gerar padrões considerados bons para o empacotamento, utilizando a técnica de geração de colunas (detalhes na Seção 3.4). Para uma base do problema dado pela Formulação (5.2), temos um vetor dual associado que denotamos por π . Na geração de colunas temos que resolver o seguinte subproblema:

$$\begin{aligned} M_k = \quad & \text{maximize} \quad \sum_{i \in I^k} \pi_i P_i \\ & \text{sob as restrições} \quad \sum_{i \in I^k} l_i P_i \leq L \\ & P_i \in \{0, 1\} \quad \forall i \in I^k \end{aligned} \tag{5.3}$$

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. Isto vem do fato que o custo reduzido é uma função das alturas e dos valores duais dos itens, ou seja, $\bar{c}_P = a_P - \pi P$, onde \bar{c}_P é o custo reduzido do padrão P . Como queremos $\bar{c}_P < 0$, basta maximizar o valor de πP resolvendo (5.3) para cada altura possível. A coluna de melhor custo reduzido deve ser devolvida para o problema mestre. Note que o número de alturas possíveis é limitado pelo número de itens. Este subproblema é chamado de Problema do Padrão de Peso Máximo.

PROBLEMA DO PADRÃO DE PESO MÁXIMO

Dados uma placa bidimensional $R = (a, b)$ e uma lista de retângulos $I = (r_1, \dots, r_n)$, onde cada retângulo $r_i = (l_i, a_i)$ tem um peso $\pi_i \in \mathbb{R}^+$, encontrar um padrão P de (R, I) de peso máximo.

No caso unidimensional, este problema recai no problema da mochila, que Vance [57] utilizou para gerar as colunas. Para a decomposição do PEBF2, podemos considerar que cada altura gera um subproblema da mochila, eliminando os itens maiores que uma determinada altura A_k antes da resolução do problema.

Embora seja NP-difícil, o problema da mochila admite um algoritmo de programação dinâmica pseudo-polinomial de tempo $O(nC)$, onde n é o número de itens e C o tamanho da mochila (vide Algoritmo 1 do Seção 4.4). Este algoritmo permite a resolução de todos subproblemas da mochila (um para cada altura possível) de uma só vez, bastando ordenar os itens em ordem crescente de altura.

Os limitantes obtidos com a Formulação (5.2) são de excelente qualidade como podemos observar nas Tabelas 6.5, 6.12 e 6.14 da Seção 6.5.

5.2 O algoritmo

Utilizando a Formulação (5.2) podemos construir um algoritmo de *branch-and-price* para obtenção de soluções exatas. O algoritmo resolve a relaxação da Formulação (5.2) utilizando a técnica de geração de colunas (que resolve, por sua vez, a Formulação (5.3)). Se a solução tiver variáveis fracionárias, devemos inserir restrições sobre estas variáveis de modo que a solução fracionária corrente seja descartada. Este processo cria subproblemas que devem ser reotimizados utilizando a técnica de geração de colunas novamente. As seções seguintes mostram os detalhes do algoritmo.

5.2.1 Ramificação

Como comentado na Seção 3.6.1, as regras de ramificação influem diretamente na geração de colunas. Portanto, vamos considerá-las primeiro e, logo após, consideraremos o problema de gerar colunas.

A forma mais simples de ramificação, já que as variáveis do Problema Mestre (5.2) são binárias, é ajustar seus limites para 0 (zero) em um ramo, digamos o esquerdo, e para 1 (um) no outro ramo, digamos direito. Seja λ_P a variável onde ajustamos os limites, então, $\lambda_P = 0$ no ramo esquerdo indica que o padrão P não pode ser usado neste ramo, e $\lambda_P = 1$ no ramo direito permite a utilização de P . Entretanto, segundo Barnhart *et al.* [1], é possível (e muito provável) que o padrão 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 de ramificação, poderemos estar em uma situação onde precisamos gerar a j -ésima melhor coluna, o que pode demandar muito tempo. Manipular o *pool* de padrões proibidos também não é uma tarefa trivial. Degraeve e Schrage [15] utilizaram esta estratégia e, sem considerar

as restrições das ramificações, tiveram que gerar a j -ésima melhor solução no j -ésimo nível da árvore.

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, aumentando sua complexidade, como já enunciado na Seção 3.6.1.

Para contornar estas dificuldades, Vance [57] propõe uma regra de ramificação baseada no trabalho de Ryan e Foster [52]. Esta regra baseia-se na seguinte proposição:

Proposição 5.2.1. *Seja X uma matriz binária e λ um vetor tal que $X\lambda = 1$ seja uma solução básica. 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 .

Demonstração. Consideremos a variável fracionária $\lambda_{k'}$. Seja r qualquer linha tal que $x_{rk'} = 1$. Como $\sum_{1 \leq k \leq p} x_{rk} \lambda_k = 1$ e $\lambda_{k'}$ é fracionária, então deve existir outra coluna básica k'' com $0 < \lambda_{k''} < 1$ e $x_{rk''} = 1$. Como não existem colunas duplicadas na base, deve existir uma linha s tal que ou $x_{sk'} = 1$ ou $x_{sk''} = 1$ mas não ambos. Assim:

$$\begin{aligned} 1 &= \sum_{1 \leq k \leq p} x_{rk} \lambda_k \\ &= \sum_{k: x_{rk}=1} \lambda_k \\ &> \sum_{k: x_{rk}=1, x_{sk}=1} \lambda_k \end{aligned}$$

onde a inequação segue do fato que o último somatório inclui ou $\lambda_{k'}$ ou $\lambda_{k''}$, mas não ambos. \square

Baseadas nesta proposição, temos as seguintes restrições para ramificação:

$$\sum_{k: x_{rk}=1, x_{sk}=1} \lambda_k = 1 \tag{5.4}$$

e

$$\sum_{k: x_{rk}=1, x_{sk}=1} \lambda_k = 0. \tag{5.5}$$

A Restrição (5.4) faz com que as linhas r e s sejam cobertas por uma mesma coluna, enquanto que a Restrição (5.5) faz com que as linhas sejam cobertas por colunas diferentes.

A Formulação (5.2) tem uma matriz binária formada por padrões P . Cada coluna representa um padrão e cada linha um item. Assim, podemos aplicar as Restrições (5.4) e (5.5) para a ramificação. No ramo cuja Restrição (5.4) é aplicada, são permitidos apenas padrões cujos itens r e s ou estejam juntos no padrão ou nenhum apareça. Note que, se a soma das larguras de r e s excedem a largura da faixa, o ramo não deve ser criado pois não poderemos gerar colunas com ambos itens. No ramo com a Restrição (5.5), apenas são permitidas as colunas onde apenas um dos itens aparece no padrão ou nenhum deles. Esta regra limita várias variáveis a cada iteração e permite que a geração de colunas seja mais fácil de ser tratada, principalmente com restrições do tipo (5.4).

A regra de ramificação é implementada da seguinte maneira. 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 tais que, em um nó, designado como filho esquerdo, o limite superior é ajustado para zero em todas as colunas que contêm um item e não contêm o outro, e no outro nó, designado como filho direito, o limite superior é ajustado para zero nas colunas que contêm ambos os itens. Esta informação é repassada para os geradores de colunas e algoritmos que calculam os limitantes superiores, através de um grafo onde os itens que devem permanecer separados são vértices ligados por uma aresta. Os itens que devem permanecer juntos, são substituídos por um novo item, resultante da combinação destes últimos (vide Seção 5.2.2). Isto garante que as 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. Por sua vez, isto implica que a relaxação de um nó filho pode ser inviável, e caso afirmativo, teremos que restaurar sua viabilidade. Para isso, utilizamos a matriz identidade eliminando as colunas que não respeitam as Restrições (5.4) e os padrões da solução gerada pelo NFDH, FFDH ou BFDH (vide Seção 5.2.3).

Utilizamos duas abordagens de escolhas dos itens. Em uma delas, dois itens quaisquer que satisfaçam a Proposição 5.2.1, podem ser escolhidos. Essa abordagem pode levar-nos 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 o grau dos vértices das componentes conexas do grafo resultante das restrições de aresta, tende a ser grande, diminuindo 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 compõem, 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 (5.7a) são resolvidos primeiro, já que sua resolução pelo algoritmo da mochila é rápida.

5.2.2 Geração de Colunas

Definidas as regras de ramificação, podemos construir os algoritmos para geração de colunas. Estes algoritmos irão resolver o problema da mochila dado pela Formulação (5.3), e suas variações causadas pela ramificação.

O problema da mochila é um dos problemas mais bem estudados, e admite resolução relativamente rápida, embora seja um problema NP-difícil. Vance [57, 58] e Vanderbeck [60] utilizaram uma variação do algoritmo Horowitz-Sahni [33], que é um algoritmo de *branch-and-bound* especializado para o problema da mochila. Nós optamos por utilizar o algoritmo de programação dinâmica apresentado na Seção 4.4, dado seu bom tempo de resposta e a capacidade de resolver múltiplos problemas da mochila de uma só vez.

A geração de colunas deve ser feita para todas alturas de níveis possíveis. No nó inicial da árvore é utilizado o algoritmo de programação dinâmica sobre a Formulação (5.3). Neste caso, a solução para todas as alturas é obtida com a tabela de apenas um problema de programação dinâmica. Mas nos nós subjacentes, devido as restrições impostas na ramificação pelas Equações (5.4) e (5.5), a geração de colunas não pode ser feita por esse algoritmo sem alterações.

As restrições impostas pela Equação (5.4) podem ser facilmente implementadas em um nó da árvore criando-se um novo item cuja largura é a soma dos itens considerados na ramificação que originou este nó, a altura é a maior entre os dois itens, e o valor é dado pela soma de seus custos reduzidos. 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:

$$\begin{aligned}
 M_k = \quad & \text{maximize} \quad \sum_{i \in S} \pi_i P_i \\
 \text{sob as restrições} \quad & \sum_{i \in S} l_i P_i \leq L \\
 & P_i \in \{0, 1\} \quad \forall i \in S
 \end{aligned} \tag{5.6}$$

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. Se o item m faz parte da solução devolvida pelo algoritmo, o padrão terá os itens r e s como componentes. Caso contrário, ambos não participarão do padrão. Note que, quanto mais restrições do tipo (5.4) forem utilizadas nas ramificações, mais os subproblemas terão tamanho menor, uma vez que substituímos dois itens por apenas um e, portanto, sua resolução será mais rápida. Observe que, se $l_r + l_s > L$, então o novo item não pode ser empacotado. Este tipo de situação deve ser controlada pelo processo de ramificação, que não deve permitir a criação de um nó deste tipo.

As restrições impostas pela Equação (5.5), que dizem que os itens devem aparecer separados, são mais complicadas. Elas remetem a seguinte formulação para cada altura A_k :

$$\begin{aligned} M_k = \quad & \text{maximize} \quad \sum_{i \in I^k} \pi_i P_i \\ & \text{sob as restrições} \quad \sum_{i \in I^k} l_i P_i \leq L \end{aligned} \tag{5.7}$$

$$\begin{aligned} P_r + P_s &\leq 1 & r, s \in I^k \\ P_i &\in \{0, 1\} & \forall i \in I^k \end{aligned} \tag{5.7a}$$

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

A Restrição (5.7a), que chamamos de *restrição de aresta*, indica que os itens r e s não devem ser empacotados no mesmo padrão. Note que, após várias ramificações, podemos ter várias restrições de aresta impostas a um nó na árvore. Denotamos por B o conjunto de pares de itens selecionados em ramificações que originam restrições de aresta, ou seja, ramificações dadas pela Equação (5.5).

O subproblema que possui restrições de aresta é resolvido por um resolvidor 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 (5.7a). Entenda-se por estas combinações, conjuntos de itens que possam ser empacotados juntos. Ambas abordagens são lentas e consomem o maior tempo do algoritmo.

O resolvidor de programação inteira, em geral, utiliza um algoritmo de enumeração implícita para resolver o problema e heurísticas associadas às estruturas peculiares do mesmo. Mesmo assim, pode gastar muito tempo na geração de uma coluna.

O teste destas combinações é custoso: embora um conjunto de itens que respeitem as restrições de aresta (5.7a) 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 (5.7a). Encontrar conjuntos de itens válidos equivale 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. Assim, 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 resolvidor para gerar a coluna. Note que se o teste das combinações devolve uma coluna com custo reduzido negativo, esta pode não ser a melhor coluna possível, já que nem toda combinação é testada.

Podemos notar que testar todas alturas possíveis nos nós com restrições de aresta, tanto utilizando o resolvidor, como testando as combinações, é muito custoso (vide resultados na Seção 6.6). Uma alternativa é devolver para a formulação mestre, a primeira coluna com custo reduzido negativo. Assim, nem todas alturas são testadas e, portanto, a coluna devolvida pode não ser a coluna ótima. Embora isso possa levar a um maior número de colunas geradas, e possivelmente ao efeito de *tailing-off* comentado na Seção 3.6.1, é efetivamente mais eficiente, como podemos observar nos resultados da Seção 6.7.

Assim, a geração de colunas que utiliza os subproblemas dados pela Formulação (5.6) é muito eficiente e traz bons resultados. Em contrapartida, a geração de colunas através dos subproblemas dados pela Formulação (5.7) é muito custosa. Na Seção 6.6 podemos ver os resultados de ambas abordagens.

5.2.3 Limitantes

Por ser um problema de minimização, o PEBF2 tem como limitante inferior a própria relaxação linear dada pela Formulação (5.2). Esse limitante é de boa qualidade, como discutido no final da Seção 5.1 e evidenciado nos resultados apresentados na Seção 6.5.

Os limitantes superiores representam soluções válidas calculadas através de heurísticas ou algoritmos aproximados. Consideramos aqui quatro algoritmos dos quais um é utilizado apenas para o cálculo do limitante superior inicial 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 o algoritmo da Mochila Gulosa apresentado na Seção 4.4. 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 (5.5).

Os limitantes superiores dos demais nós da árvore são calculados pelos algoritmos NFDH, FFDH ou BFDH (vide Capítulo 4). Mas para isso, estes algoritmos devem ser modificados para atenderem as restrições de aresta (5.7a). Esta modificação consiste simplesmente em verificar a compatibilidade do item a ser empacotado com os itens do nível tratado no momento. Ou seja, antes de ser empacotado em um nível, o item é testado contra todos os outros itens pertencentes ao nível, procurando por alguma violação na restrição de aresta. Se não ocorrer tal violação, o item é empacotado neste nível. Caso contrário, procede-se da mesma maneira quando o item não cabe no nível. Esta modificação faz os algoritmos terem complexidade de tempo $O(n^2)$ e pode não garantir os fatores de aproximação do NFDH e FFDH.

5.3 Implementação

A implementação de um algoritmo de *branch-and-price* envolve uma série de questões essenciais para o desempenho deste. É preciso um gerenciamento eficiente da árvore, controlando a quantidade de informação presente nesta para que não tenhamos gargalo de memória por replicação desnecessária de informação, nem latência no processamento de informações reduzidas. Isto significa controlar os nós ativos, as colunas necessárias em cada nó, as informações referentes a ramificação e geração de colunas.

A implementação de todas estas funcionalidades, além das básicas como ramificação, poda e geração de colunas, não é uma tarefa trivial. Existem vários *frameworks* que auxiliam na construção destes algoritmos. Neste trabalho utilizamos o *framework* BCP da infra-estrutura COIN [10].

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. Sua estrutura permite o acoplamento de uma grande variedade de resolvedores, tornando a implementação independente destes. O BCP foi concebido para a implementação em paralelo. Neste trabalho, não utilizamos tal recurso devido a licença do resolvedor utilizado e a disponibilidade de máquinas paralelas.

Infelizmente, o BCP apresenta uma documentação deficiente, tanto na descrição de seus componentes, como nos exemplos ilustrativos. A maioria deles está concentrada na geração de cortes. Inclusive, no documento principal que o apresenta e o descreve, há uma clara referência sobre a falta de informações sobre a geração de colunas. Isto tornou a implementação mais lenta que o esperado. Algumas alterações no código-

fonte do BCP e OSI (componente da COIN que provê interfaces para diversos resolveadores) tiveram que ser feitas para que estes funcionassem corretamente com a geração de colunas, principalmente no que consta a interface para o resolvedor Xpress-MP® [67]. Nos principais trechos de código que reportam erros existem comentários do tipo “Conserte-me” (*FIXME*).

O BCP é dividido em dois módulos principais: um gerenciador da árvore de ramificação chamado TM (*Tree Manager*), e outro gerenciador do resolvedor chamado LP (*Linear Program Manager*). Existem ainda os módulos de geração de corte CG (*Cut Generator*), e geração de colunas VG (*Variable Generator*), concebidos para utilização em sistemas paralelos/distribuídos. Estes dois últimos módulos não foram utilizados.

O módulo TM é responsável pela manutenção dos nós da árvore, cada qual com seu conjunto de restrições e variáveis. São mantidas apenas uma cópia das restrições e variáveis comuns a todos os nós, que são chamados de objetos de núcleo, e cada nó mantém seu conjunto específico. Em nosso trabalho, apenas temos restrições como objetos de núcleo. Como geramos as variáveis, as mesmas recebem o nome de objetos algorítmicos e uma cópia é armazenada em cada nó da árvore. O módulo TM é responsável também pela decisão sobre a seqüência de processamento dos nós. Procuramos sempre processar primeiramente nós que não tenham restrições de arestas (5.7a), pois seu processamento é mais rápido (vide Seção 5.2.2). A escolha entre nós do mesmo tipo é feita por um algoritmo interno do COIN, onde podemos ajustar algumas preferências como busca em profundidade, busca em largura ou busca baseada em melhores limitantes. É no TM também que ocorre a leitura dos dados dos usuários e a inicialização das estruturas de dados e resolvedor.

O módulo LP é responsável pela resolução das relaxações lineares, pelos processos de geração de coluna, geração de soluções heurísticas, cálculo do limitante superior e inferior, e ramificação. No início do processamento de cada nó é gerada uma solução heurística utilizando um dos algoritmos mostrados na Seção 5.2.3, que é devolvida ao TM para atualização do limitante superior. O laço principal inicia-se com a resolução da relaxação. Se uma solução inteira é encontrada, a mesma é devolvida para o TM. Caso contrário, segue-se a geração de colunas. Caso seja encontrada uma coluna com o custo reduzido negativo, esta é inserida na formulação e o sistema reotimizado. Caso contrário, realiza-se a ramificação, onde são construídas as informações para a geração de colunas válidas nos dois filhos do nó corrente. Caso não seja possível realizar a ramificação, o LP informa o TM que o nó corrente deve ser podado. O limitante inferior é obtido através da relaxação do nó raiz quando não conseguimos gerar mais colunas.

Capítulo 6

Experimentos Computacionais

6.1 Ambiente Computacional

Todos os algoritmos propostos foram implementados na linguagem C++ e compilados com o compilador GNU G++ (GCC) 4.0.3, sobre o sistema Debian GNU/Linux, kernel 2.6.10.

O resolvidor de programação linear inteira utilizado foi o Xpress-MP® [67], versão 16.10.03¹. Este é um resolvidor comercial, com grande visão no mercado. Embora existam resolvidores cujo código é livre, como o GNU GLPK [30], o Xpress foi escolhido devido sua robustez na resolução de programas lineares inteiros na geração de colunas (Seção 5.2.2) e a disponibilidade de licenças.

A máquina utilizada, na maioria dos ensaios, foi um Pentium® 4 2GHz com 1GB de memória RAM. Também foi utilizada como apoio, uma máquina Pentium® 4 HT 3.2GHz com 2GB de memória RAM. Os resultados apresentados aqui são todos referentes a primeira máquina, salvo referência explícita à segunda.

6.2 Instâncias Testadas

As instâncias de teste foram adaptadas do repositório ORLIB [48] que contém apenas algumas instâncias para o problema de empacotamento bidimensional em faixas PEBF (sem restrições de guilhotina). Estas instâncias, nomeadas de *ht*, são originárias do trabalho de Hopper e Turton [32]. Utilizamos outras instâncias da ORLIB propostas para o problema de empacotamento bidimensional e o problema de corte bidimensional. Neste caso, a largura da faixa corresponde a largura dos recipientes destas instâncias. Utilizamos as instâncias descritas por Christofides e Whitlock [7], chamadas de *gccut*,

¹Embora pareça, a representação da versão do Xpress não é uma data.

e instâncias descritas por Beasley [3], chamadas de *gcut*. As instâncias descritas por Bengtsson [5], chamadas de *beng*, foram sugeridas pelo trabalho de Martello *et al.* [45] e não constam na ORLIB. O número de instâncias de teste é de 47.

6.3 Verificação do algoritmo

Para comprovar a corretude do algoritmo, foram utilizadas instâncias do problema do empacotamento unidimensional PEU, considerando a mesma altura para os itens. Assim, o número de níveis gerados é igual ao número de recipientes no PEU. Em particular, utilizamos algumas instâncias propostas por Falkenauer [22], cujos valores de soluções ótimas são conhecidas. Estas instâncias podem ser encontradas no repositório ORLIB.

As Tabelas 6.1 e 6.2 mostram os resultados para 20 instâncias testadas. A Tabela 6.1 é formada pelas colunas: *Nome*, que descreve o nome da instância; *Itens*, o número de itens da instância; *Obj.*, que descreve o valor da melhor solução encontrada; *OPT?*, que indica se uma solução ótima foi encontrada, através de um *. A coluna *Nós* representa o número de nós: *Liv.*, sem restrições de aresta; *Res.*, com restrições de aresta; *Total*, total de nós; e *Prof.*, profundidade da árvore de ramificação.

Tabela 6.1: Resultados das instâncias do PEU: características da árvore.

Nome	Itens	Obj.	Opt?	Nós			
				Liv.	Res.	Total	Prof.
u120_00	120	48	*	44	43	87	43
u120_01	120	49	*	34	33	67	33
u120_02	120	46	*	51	50	101	50
u120_03	120	49	*	36	35	71	35
u120_04	120	50	*	44	43	87	43
u120_05	120	48	*	58	57	115	57
u120_06	120	48	*	55	54	109	54
u120_07	120	49	*	52	51	103	51
u120_08	120	50	*	32	31	63	31
u120_09	120	46	*	35	34	69	34
u120_10	120	52	*	31	30	61	30
u120_11	120	49	*	50	49	99	49
u120_12	120	48	*	38	37	75	37
u120_13	120	49	*	1	0	1	0
u120_14	120	50	*	47	46	93	46
u120_15	120	48	*	55	54	109	54
u120_16	120	52	*	32	31	63	31
u120_17	120	52	*	35	34	69	34
u120_18	120	49	*	49	48	97	48
u120_19	120	49	*	29	28	57	28

A Tabela 6.2 mostra informações referentes à geração de colunas e tempo de execução. Os campos *Nome* e *Itens* são idênticos os da tabela anterior. A seção *Variáveis* é dividida nas colunas: *Moc*, que indica o número de variáveis geradas nos nós sem restrições de arestas; *PLI* e *Comb*, as variáveis geradas pelo resolvidor e pelo teste das combinações, respectivamente; *Total*, o número total de variáveis geradas. A seção *Geração de Colunas* indica o tempo, em segundos de CPU, da geração de colunas. A descrição das colunas é idêntica à descrição da seção anterior. A coluna *Heur.* indica o tempo gasto na geração dos limitantes superiores, *Ram.*, o tempo gasto na escolha de itens para a ramificação e, *Total*, o tempo total de execução do algoritmo.

Tabela 6.2: Resultados das instâncias do PEU: geração de colunas.

Nome	Itens	Variáveis				Geração de colunas (s)				Tempo (s)		
		Moc.	PLI	Comb.	Total	Moc.	PLI	Comb.	Total	Heur.	Ram.	Total
u120_00	120	391	2	29	422	0.30	2.82	0.52	3.77	0.01	0.19	13.16
u120_01	120	384	1	27	412	0.29	0.41	0.42	1.27	0.02	0.14	10.39
u120_02	120	450	3	35	488	0.33	0.94	0.58	2.00	0.03	0.18	12.76
u120_03	120	408	4	25	437	0.28	1.00	0.42	1.84	0.01	0.18	10.78
u120_04	120	386	4	31	421	0.28	0.28	0.50	1.20	0.00	0.16	9.86
u120_05	120	437	7	35	479	0.34	1.10	0.60	2.17	0.01	0.24	12.58
u120_06	120	459	4	42	505	0.31	0.82	0.61	1.88	0.02	0.22	13.70
u120_07	120	397	3	36	436	0.30	2.02	0.57	3.09	0.02	0.18	11.83
u120_08	120	363	1	27	391	0.32	0.63	0.42	1.48	0.00	0.14	9.19
u120_09	120	484	2	28	514	0.34	0.50	0.43	1.44	0.00	0.17	13.27
u120_10	120	331	0	28	359	0.25	0.19	0.37	0.90	0.01	0.12	7.99
u120_11	120	348	6	28	382	0.26	2.64	0.56	3.59	0.01	0.19	11.46
u120_12	120	402	2	30	434	0.25	0.53	0.47	1.35	0.01	0.13	9.51
u120_13	120	192	0	0	192	0.18	0.00	0.00	0.22	0.00	0.00	3.48
u120_14	120	435	8	31	474	0.30	2.12	0.54	3.10	0.01	0.22	13.52
u120_15	120	489	6	35	530	0.37	2.79	0.62	3.96	0.04	0.22	15.52
u120_16	120	349	0	27	376	0.30	0.38	0.38	1.15	0.02	0.13	8.43
u120_17	120	298	5	23	326	0.15	0.24	0.42	0.88	0.02	0.15	6.50
u120_18	120	479	3	37	519	0.33	2.93	0.54	3.96	0.01	0.22	15.43
u120_19	120	344	4	22	370	0.28	0.07	0.35	0.82	0.01	0.10	8.4

Os testes nesta fase, além de demonstrar a corretude, serviram para confirmar os resultados de Vance [57] no caso do PEU, onde há uma tendência do algoritmo a pesquisar nós mais à esquerda (onde geramos variáveis pela programação dinâmica da mochila).

6.4 Limitantes Superiores

Primeiramente, avaliamos os algoritmos para cálculo dos limitantes superiores e os resultados podem ser vistos na Tabela 6.3. As colunas estão dispostas da seguinte maneira: a coluna *Instância* indica o nome da instância; a coluna *Itens*, o número de itens da instância; as seções *NFDH*, *FFDH*, *BFDH* e *MG* da tabela contém as colunas *Obj.* e *Tempo* indicando, respectivamente, o valor da solução encontrada e o tempo

gasto, em milissegundos de CPU, pelos algoritmos NFDH, FFDH, BFDH e mochila gulosa. Os valores em negrito são os melhores encontrados para uma instância dentro dos valores gerados pelos algoritmos. Nas linhas onde nenhuma valor foi assinalado, os algoritmos encontraram soluções de mesmo valor. Estes ensaios foram realizados na máquina de apoio descrita na Seção 6.1.

Tabela 6.3: Comparação dos algoritmos para geração do limitante superior. Tempo em milissegundos (ms).

Instância	Items	NFDH		FFDH		BFDH		MG	
		Obj.	Tempo	Obj.	Tempo	Obj.	Tempo	Obj.	Tempo
beng01	20	36	0.03	36	0.06	36	0.10	36	0.18
beng02	40	70	0.05	62	0.12	62	0.18	62	0.40
beng03	60	99	0.08	89	0.17	89	0.27	88	0.70
beng04	80	124	0.11	115	0.23	114	0.36	113	0.15
beng05	100	157	0.13	142	0.29	142	0.46	138	0.64
beng06	40	45	0.05	40	0.11	40	0.17	41	0.45
beng07	80	76	0.10	72	0.22	71	0.34	71	0.32
beng08	120	115	0.15	105	0.32	105	0.50	105	0.50
beng09	160	140	0.21	130	0.43	129	0.67	129	0.10
beng10	200	176	0.25	160	0.54	160	0.84	159	0.03
cgcut1	7	14	0.01	14	0.03	14	0.04	14	0.07
cgcut2	10	51	0.02	51	0.04	51	0.06	53	0.11
cgcut3	20	254	0.03	230	0.06	230	0.10	232	0.22
gcut01	10	1016	0.02	1016	0.04	1016	0.06	1016	0.13
gcut02	20	1564	0.03	1349	0.07	1349	0.10	1347	0.34
gcut03	30	1971	0.04	1873	0.09	1810	0.14	1899	0.65
gcut04	50	3992	0.07	3216	0.15	3216	0.23	3159	0.54
gcut05	10	1572	0.02	1572	0.04	1572	0.06	1375	0.21
gcut06	20	3565	0.03	3139	0.07	3139	0.10	2862	0.48
gcut07	30	6443	0.04	5188	0.10	5151	0.15	4979	0.76
gcut08	50	7792	0.07	6728	0.15	6496	0.23	6301	0.42
gcut09	10	3363	0.02	2664	0.04	2646	0.06	2664	0.23
gcut10	20	8208	0.03	6554	0.07	6554	0.11	6539	0.88
gcut11	30	8570	0.04	7610	0.09	7604	0.14	7571	0.88
gcut12	50	18356	0.07	15827	0.16	15827	0.25	15266	0.70
gcut13	32	6519	0.04	5500	0.09	5415	0.14	5500	4.28
ht_c1_01	16	31	0.02	27	0.05	27	0.08	28	0.14
ht_c1_02	17	31	0.03	30	0.05	30	0.08	29	0.14
ht_c1_03	16	23	0.02	23	0.05	23	0.08	23	0.13
ht_c2_01	25	20	0.03	20	0.07	20	0.11	22	0.20
ht_c2_02	25	35	0.03	34	0.07	34	0.11	34	0.20
ht_c2_03	25	23	0.04	23	0.07	23	0.11	23	0.20

Continua na próxima página...

Tabela 6.3 – Continuação

Instância	Items	NFDH		FFDH		BFDH		MG	
		Obj.	Tempo	Obj.	Tempo	Obj.	Tempo	Obj.	Tempo
ht_c3_01	28	40	0.04	40	0.08	40	0.12	41	0.26
ht_c3_02	29	42	0.04	42	0.08	42	0.13	42	0.28
ht_c3_03	28	43	0.04	43	0.08	43	0.12	43	0.27
ht_c4_01	49	79	0.06	74	0.13	74	0.21	74	0.57
ht_c4_02	49	80	0.06	74	0.14	74	0.21	74	0.60
ht_c4_03	49	89	0.06	80	0.14	80	0.21	81	0.56
ht_c5_01	73	105	0.09	101	0.20	101	0.31	102	0.13
ht_c5_02	73	120	0.10	106	0.20	106	0.32	106	0.05
ht_c5_03	73	123	0.09	107	0.20	107	0.32	107	0.13
ht_c6_01	97	144	0.12	136	0.26	136	0.40	136	0.96
ht_c6_02	97	162	0.12	147	0.27	145	0.41	149	0.08
ht_c6_03	97	151	0.12	139	0.26	139	0.41	141	0.92
ht_c7_01	196	283	0.26	261	0.55	261	0.85	262	0.90
ht_c7_02	197	303	0.26	283	0.56	283	0.87	283	4.29
ht_c7_03	196	284	0.26	273	0.54	273	0.83	274	0.33

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 47 instâncias (29,79%), nenhuma melhoria em 18 delas (38,30%) e perdeu em 15 (31,91%). Tanto o BFDH como o algoritmo da mochila gulosa apresentaram tempo de execução muito bons, raramente passando da casa de 1 milissegundo.

As Figuras 6.1 e 6.2 mostram a comparação entre os algoritmos para algumas instâncias. As instâncias da Figura 6.1 são difíceis pois algoritmo de *branch-and-price* encontrou uma solução ótima em apenas 2 delas. As instâncias da Figura 6.2 são numericamente grandes. As dimensões dos itens e faixa destas últimas estão na casa das centenas e milhares, enquanto as dimensões de todas outras instâncias flutuam na casa das dezenas, raramente na casa das centenas. O eixo horizontal mostra o nome das instâncias e o eixo vertical os valores das soluções encontradas normalizados, pela pior solução. Não apresentamos os resultados do NFDH, já que apresentou os piores resultados em todas as instâncias.

Aplicamos a depreciação baseada na largura do item em relação a capacidade da mochila, no algoritmo da mochila gulosa. Utilizamos a relação de 5%, 10%, 15%, 20%, 25% e 30% nas funções descritas na Seção 4.4. Em apenas uma instância (beng04) obtivemos melhoria. Portanto, optamos em não utilizar as funções de amortização.

A Tabela 6.4 mostra o *gap* entre o valor da solução ótima e o valor da solução encontrada por cada algoritmo. As colunas *Instância* e *Items* indicam o nome e o número

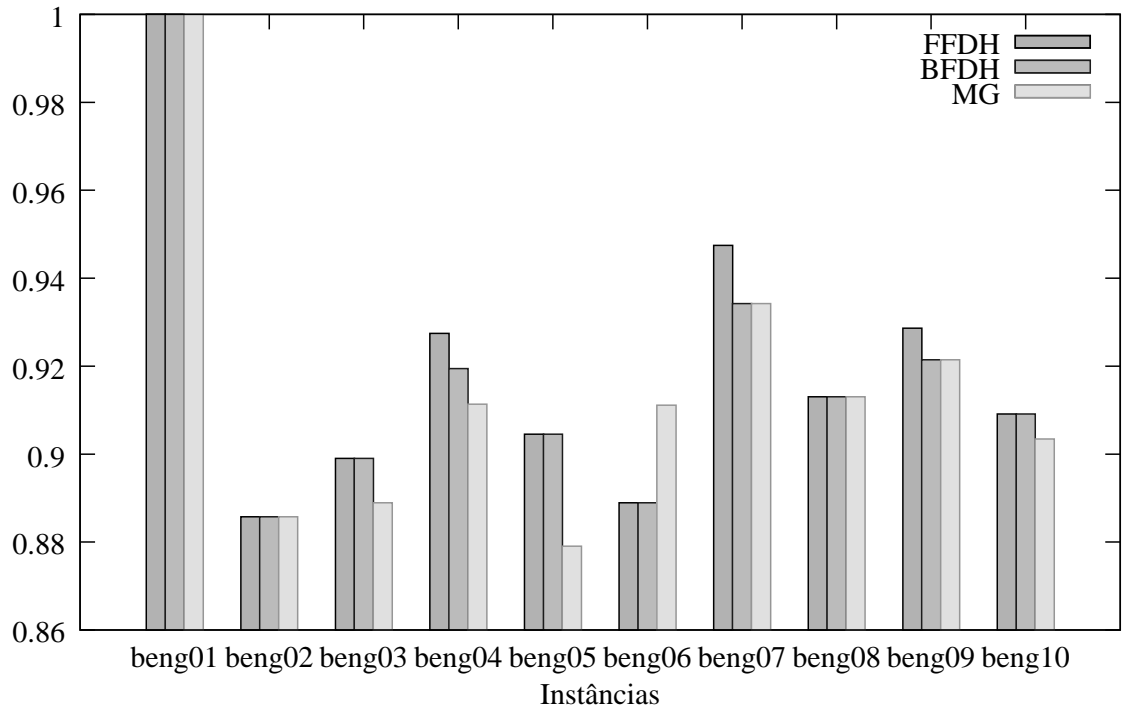


Figura 6.1: Limitantes para instâncias de beng01 a beng10.

de itens da instância, respectivamente; a coluna *Ótimo* indica o valor da solução ótima encontrada; e as colunas incluídas na seção *Algoritmos* da tabela representam o *gap* do valor da solução encontrada pelo algoritmo e o valor da solução ótima. Os melhores valores estão em negrito. Sejam $OPT(I)$ o valor da solução ótima e $A(I)$ o valor da solução do algoritmo. O *gap* é calculado através da equação

$$Gap = 100 \cdot \frac{A(I) - OPT(I)}{OPT(I)}.$$

Note que somente as instâncias onde uma solução ótima foi encontrada, estão presentes na tabela, totalizando 26 das 47 instâncias testadas. O NFDH devolveu uma solução ótima em 10 casos e seu *gap* médio foi de 11,16%. O FFDH apresentou uma solução ótima em 13 casos com *gap* médio de 2,85%. O BFDH devolveu 15 soluções ótimas e teve 2,50% de *gap* médio. Por fim, o algoritmo da mochila gulosa alcançou 11 soluções ótimas mas teve *gap* médio inferior aos outros algoritmos: 2,01%.

O algoritmo da mochila gulosa obteve um bom desempenho nas instâncias *gcut*. A hipótese mais provável é que, por considerar a área dos itens, o algoritmo da mochila gulosa pode aproveitar mais as informações dos itens destas instâncias, já que são numericamente maiores que os itens das demais instâncias. O algoritmo alcançou uma solução ótima em 3 casos (*gcut01*, *gcut06* e *gcut07* marcadas com [†] na Tabela

Tabela 6.4: *Gap* entre o valor da solução ótima e o valor da solução de cada algoritmo.

Instância	Items	Ótimo	Algoritmos				
			NFDH	FFDH	BFDH	MG	
beng01	20	36	0,00	0,00	0,00	0,00	
beng02	40	61	14,75	1,64	1,64	1,64	
cgcut1	7	14	0,00	0,00	0,00	0,00	
cgcut2	10	51	0,00	0,00	0,00	3,92	
cgcut3	20	230	10,43	0,00	0,00	0,87	
gcut01	10	1016	0,00	0,00	0,00	0,00	† *
gcut02	20	1262	23,93	6,89	6,89	6,74	
gcut03	30	1810	8,90	3,48	0,00	4,92	*
gcut05	10	1360	15,59	15,59	15,59	1,10	
gcut06	20	2862	24,56	9,68	9,68	0,00	†
gcut07	30	4979	29,40	4,20	3,45	0,00	†
gcut08	50	6168	26,33	9,08	5,32	2,16	
gcut09	10	2646	27,10	0,68	0,00	0,68	*
gcut10	20	6167	33,10	6,28	6,28	6,03	
gcut11	30	7298	17,43	4,28	4,19	3,74	
gcut12	50	14943	22,84	5,92	5,92	2,16	
ht_c1_01	16	27	14,81	0,00	0,00	3,70	
ht_c1_02	17	29	6,90	3,45	3,45	0,00	
ht_c1_03	16	23	0,00	0,00	0,00	0,00	
ht_c2_01	25	20	0,00	0,00	0,00	10,00	
ht_c2_02	25	34	2,94	0,00	0,00	0,00	
ht_c2_03	25	23	0,00	0,00	0,00	0,00	
ht_c3_01	28	40	0,00	0,00	0,00	2,50	
ht_c3_02	29	42	0,00	0,00	0,00	0,00	
ht_c3_03	28	43	0,00	0,00	0,00	0,00	
<i>Gap</i> médio			11,16	2,85	2,50	2,01	

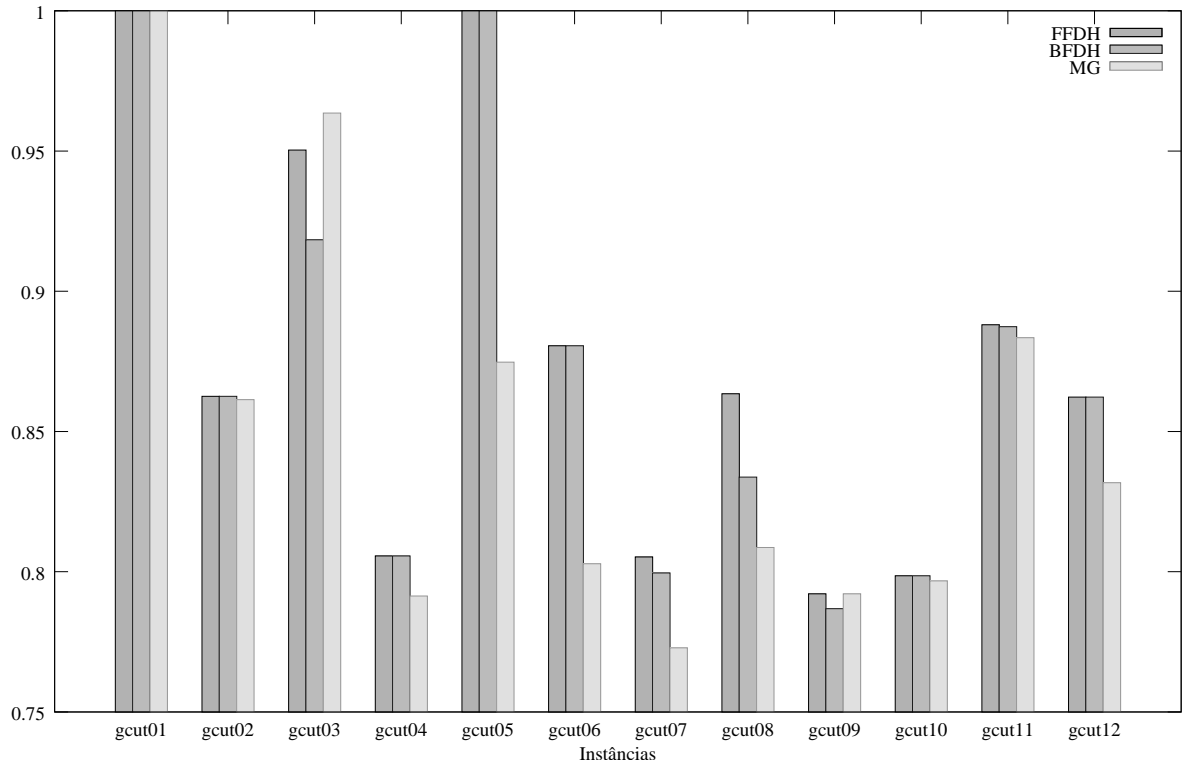


Figura 6.2: Limitantes para instâncias de gcut01 a gcut12.

6.4) com o *gap* médio de 2,29%, destas instâncias. O BFDH também alcançou 3 soluções ótimas (gcut01, gcut03 e gcut09 marcadas com * na Tabela 6.4) mas seu *gap* médio foi pior nestas instâncias: 4,78%.

6.5 Resultados Iniciais

Dados os resultados da seção anterior, optamos por aplicar o algoritmo da mochila gulosa sem amortização para gerar o limitante superior inicial e o algoritmo BFDH para gerar os limitantes superiores nos demais nós. A geração de colunas em nós com restrições de aresta foi feita pelo resolvidor que devolve a primeira coluna com custo reduzido negativo. As Tabelas 6.5 e 6.6 mostram os resultados do algoritmo de *branch-and-price* para o PEBF2.

A Tabela 6.5 é composta pelas colunas Nome, que descreve o nome da instância, e Itens que descreve o número de itens da instância. A seção Limitantes mostra o valor dos limitantes: P. Inf., primeiro limitante inferior; M. Inf., melhor limitante inferior; Sup., melhor limitante superior. A coluna Obj. descreve o valor da melhor solução encontrada em 3600 segundos de CPU; Dif. descreve a diferença, em unida-

des, entre limitante inferior e o valor objetivo; *Gap* descreve o *gap* proporcional entre o melhor limitante inferior e o valor da melhor solução encontrada; *Opt?* indica se uma solução ótima foi encontrada ou não. A seção *Nós* representa o número de nós: *Liv.*, livre de restrições de aresta; *Res.*, com restrições de aresta; *Total*, total de nós. A coluna *Prof.* indica a profundidade da árvore.

O *gap* é calculado através da equação

$$Gap = 100 \cdot \frac{Obj - M.Inf.}{Obj}.$$

O *gap* médio obtido foi de 1,35%, com desvio padrão 1,68, nas 47 instâncias testadas. Dentre estas, uma solução ótima foi encontrada em 24 instâncias (51,06%). Entre as 23 instâncias (48,94%) onde nenhuma solução ótima foi encontrada, o *gap* médio foi de 2,75%, com desvio padrão 1,35. Estes valores atestam a qualidade da formulação quanto à obtenção de limitantes inferiores.

A Tabela 6.6 mostra os resultados quanto a geração de colunas e tempo gasto em cada parte do algoritmo. É composta pelas colunas *Nome* e *Itens*, como descritas anteriormente. A seção *Variáveis* é dividida nas colunas: *Moc.*, que indica o número de variáveis geradas nos nós sem restrições de arestas; *PLI*, as variáveis geradas pelo resolvidor, e *Total*, o número total de variáveis geradas. A seção *Geração de Colunas* indica o tempo, em segundos de CPU, da geração de colunas. A descrição das colunas é idêntica à descrição da seção anterior. A seção *Tempo* indica o tempo, em segundos de CPU, das operações: *Heur.*, geração dos limitantes superiores, *Ram.*, ramificações e *Total*, o tempo total de execução do algoritmo. O tempo máximo de execução foi ajustado em 3600 segundos (1 hora) de CPU.

Infelizmente, o algoritmo não se comportou da maneira esperada quando tratou as instâncias do PEBF2. Na Seção 6.3, observamos que, para o PEU, o algoritmo gerou árvores onde os nós sem restrições de aresta são em maior número, que é uma vantagem para geração de colunas. No caso do PEBF2, as árvores geradas foram mais balanceadas ou tenderam a crescer com nós com restrições de aresta. Isto pode ser comprovado pela Figura 6.3 da instância *cgcut2*. Cada elipse representa um nó da árvore de ramificação identificado por um número que indica sua ordem de criação. O nó é do tipo *Liv* se não contém restrições de aresta, e do tipo *PLI* caso contrário, e contém as informações *LI* indicando o limitante inferior daquele nó, e *Itens* indicando os itens que participam da ramificação. A elipse preenchida representa o nó onde a primeira solução ótima foi encontrada.

Desta maneira, o número de nós com restrições de aresta foi muito maior que o número de nós sem estas. Na maioria dos casos, enquanto o número de nós sem restrições de aresta flutuou na casa das dezenas, o número de nós com estas restrições

Tabela 6.5: Resultados utilizando o algoritmo da mochila gulosa e BFDH: características da árvore.

Nome	Itens	Limitantes			Obj.	Dif.	Gap	Opt?	Nós			
		P. Inf.	M. Inf.	Sup.					Liv.	Res.	Total	Prof.
beng01	20	33	36	36	36	0	0.00	*	4	4637	4641	28
beng02	40	60	61	61	61	0	0.00	*	11	5882	5893	23
beng03	60	86	86	88	88	2	2.33		18	15845	15863	38
beng04	80	108	108	113	112	4	3.70		39	10368	10407	38
beng05	100	134	134	138	138	4	2.99		36	8969	9005	37
beng06	40	37	38	40	40	2	5.26		22	13707	13729	27
beng07	80	68	68	71	70	2	2.94		49	5744	5793	48
beng08	120	101	101	105	105	4	3.96		75	4220	4295	74
beng09	160	126	126	129	129	3	2.38		83	3518	3601	92
beng10	200	156	156	159	159	3	1.92		9	1760	1769	125
cgcut1	7	12	14	14	14	0	0.00	*	3	2	5	2
cgcut2	10	46	51	51	51	0	0.00	*	4	49	53	8
cgcut3	20	220	228	230	229	1	0.44		8	38463	38471	32
gcut01	10	1016	1016	1016	1016	0	0.00	*	1	0	1	0
gcut02	20	1262	1262	1262	1262	0	0.00	*	8	11	19	7
gcut03	30	1810	1810	1810	1810	0	0.00	*	1	0	1	0
gcut04	50	3108	3123	3159	3126	3	0.10		4	5003	5007	24
gcut05	10	1360	1360	1360	1360	0	0.00	*	1	0	1	0
gcut06	20	2792	2862	2862	2862	0	0.00	*	7	126	133	10
gcut07	30	4894	4979	4979	4979	0	0.00	*	4	1103	1107	18
gcut08	50	6149	6168	6168	6168	0	0.00	*	20	479	499	19
gcut09	10	2509	2646	2646	2646	0	0.00	*	4	27	31	6
gcut10	20	6167	6167	6167	6167	0	0.00	*	1	0	1	0
gcut11	30	7298	7298	7298	7298	0	0.00	*	1	0	1	0
gcut12	50	14943	14943	14943	14943	0	0.00	*	1	0	1	0
gcut13	32	5090	5103	5415	5398	295	5.78		20	3175	3195	27
ht_c1_01	16	26	27	27	27	0	0.00	*	2	1	3	1
ht_c1_02	17	29	29	29	29	0	0.00	*	1	0	1	0
ht_c1_03	16	23	23	23	23	0	0.00	*	1	0	1	0
ht_c2_01	25	20	20	20	20	0	0.00	*	1	0	1	0
ht_c2_02	25	33	34	34	34	0	0.00	*	3	18	21	9
ht_c2_03	25	23	23	23	23	0	0.00	*	1	0	1	0
ht_c3_01	28	37	40	40	40	0	0.00	*	10	109	119	11
ht_c3_02	29	41	42	42	42	0	0.00	*	2	1	3	1
ht_c3_03	28	42	43	43	43	0	0.00	*	4	3	7	3
ht_c4_01	49	71	73	74	74	1	1.37		15	2844	2859	35
ht_c4_02	49	73	73	74	74	1	1.37		9	790	799	29
ht_c4_03	49	72	73	75	75	2	2.74		17	2745	2762	27
ht_c5_01	73	103	104	107	107	3	2.88		25	2102	2127	29
ht_c5_02	73	104	104	106	106	2	1.92		7	756	763	26
ht_c5_03	73	103	103	107	106	3	2.91		29	1976	2005	28
ht_c6_01	97	131	132	136	136	4	3.03		22	723	745	37
ht_c6_02	97	139	140	145	145	5	3.57		26	697	723	36
ht_c6_03	97	137	137	139	139	2	1.46		20	575	595	39
ht_c7_01	196	254	254	261	261	7	2.76		99	154	253	98
ht_c7_02	197	272	272	283	282	10	3.68		71	110	181	70
ht_c7_03	196	263	263	273	273	10	3.80		112	127	239	111

Tabela 6.6: Resultados utilizando o algoritmo da mochila gulosa e BFDH: geração de colunas.

Nome	Itens	Variáveis			Geração de colunas (s)			Tempo (s)		
		Moc.	PLI	Total	Moc.	PLI	Total	Heur.	Ram.	Total
beng01	20	46	14706	14752	0.01	229.15	230.32	0.56	3.12	282.70
beng02	40	120	49519	49639	0.03	1394.24	1398.91	1.13	7.15	1661.04
beng03	60	238	61101	61339	0.09	3081.98	3091.23	2.40	24.54	3600.03
beng04	80	429	40355	40784	0.19	2905.98	2913.73	2.06	22.68	3600.43
beng05	100	481	35493	35974	0.30	2862.09	2870.53	2.05	22.30	3600.33
beng06	40	352	84717	85069	0.13	2988.51	2999.20	1.34	19.73	3600.90
beng07	80	724	45990	46714	0.47	2846.43	2855.38	0.94	15.24	3600.80
beng08	120	1080	38055	39135	0.99	2388.55	2399.66	0.99	17.27	3600.36
beng09	160	1561	31745	33306	1.92	2036.43	2050.63	0.94	23.33	3602.92
beng10	200	1221	19788	21009	2.00	1254.06	1266.07	0.60	18.85	3600.88
cgcut1	7	11	0	11	0.00	0.01	0.01	0.00	0.00	0.03
cgcut2	10	26	155	181	0.00	1.49	1.49	0.00	0.03	1.95
cgcut3	20	53	66575	66628	0.03	3310.93	3316.80	3.74	24.98	3600.03
gcut01	10	0	0	0	0.00	0.00	0.00	0.00	0.00	0.01
gcut02	20	34	12	46	0.04	0.39	0.43	0.00	0.01	0.57
gcut03	30	2	0	2	0.01	0.00	0.01	0.00	0.00	0.02
gcut04	50	96	11351	11447	0.74	3518.08	3519.99	0.75	5.84	3600.71
gcut05	10	8	0	8	0.01	0.00	0.01	0.00	0.00	0.03
gcut06	20	33	193	226	0.09	8.98	9.08	0.02	0.07	9.97
gcut07	30	29	1298	1327	0.16	154.78	155.07	0.15	0.73	163.13
gcut08	50	144	1966	2110	1.92	640.27	642.39	0.12	0.59	655.45
gcut09	10	12	16	28	0.02	0.58	0.61	0.00	0.01	0.74
gcut10	20	14	0	14	0.07	0.00	0.07	0.00	0.00	0.11
gcut11	30	42	0	42	0.49	0.00	0.49	0.00	0.00	0.60
gcut12	50	24	0	24	0.76	0.00	0.76	0.00	0.00	0.83
gcut13	32	328	28798	29126	6.71	3416.46	3426.17	0.31	4.65	3601.06
ht_c1_01	16	31	15	46	0.00	0.18	0.19	0.00	0.00	0.28
ht_c1_02	17	46	0	46	0.01	0.00	0.01	0.00	0.00	0.10
ht_c1_03	16	94	0	94	0.01	0.00	0.01	0.00	0.00	0.22
ht_c2_01	25	333	0	333	0.11	0.00	0.13	0.00	0.00	2.57
ht_c2_02	25	271	704	975	0.13	5.88	6.09	0.01	0.03	12.96
ht_c2_03	25	223	0	223	0.08	0.00	0.10	0.00	0.00	1.43
ht_c3_01	28	250	5133	5383	0.14	97.63	98.29	0.03	0.19	136.15
ht_c3_02	29	297	158	455	0.31	4.61	4.96	0.00	0.01	9.01
ht_c3_03	28	280	553	833	0.19	9.71	9.97	0.00	0.01	17.06
ht_c4_01	49	449	45733	46182	0.53	3055.56	3061.70	0.45	6.60	3607.95
ht_c4_02	49	525	38410	38935	0.77	2981.97	2987.88	0.21	2.67	3601.66
ht_c4_03	49	478	31931	32409	0.63	2978.54	2979.17	0.57	5.32	3605.31
ht_c5_01	73	934	21818	22752	7.78	2993.27	3001.05	0.22	5.45	3607.37
ht_c5_02	73	1207	18816	20023	3.19	2861.77	2869.11	0.16	4.42	3608.42
ht_c5_03	73	831	21917	22748	7.49	2893.72	2901.21	0.37	5.73	3611.48
ht_c6_01	97	1487	14561	16048	7.98	2693.77	2701.75	0.19	5.98	3607.67
ht_c6_02	97	1421	13582	15003	7.85	2515.30	2523.15	0.21	6.13	3615.72
ht_c6_03	97	1555	17462	19017	6.54	2530.64	2541.91	0.14	5.20	3601.49
ht_c7_01	196	4487	1443	5930	80.12	2625.01	2707.64	0.09	74.15	3703.50
ht_c7_02	197	5970	1159	7129	172.57	1088.07	1264.44	0.08	186.72	3626.62
ht_c7_03	196	4977	2653	7630	116.01	3863.92	3983.50	0.10	67.87	4054.57

oscilou na casa dos milhares. O tempo total gasto na geração de colunas nesses nós foi diretamente proporcional à sua quantidade.

O grande número de nós com restrições de aresta aumentou consideravelmente o tempo de processamento, impossibilitando a resolução, na otimalidade, de instâncias com mais de 50 itens, salvo as instâncias `gcut08` e `gcut12`. Portanto, modificações na geração de colunas foram feitas, na tentativa de acelerar o processo de geração, e seus resultados são apresentados na próxima seção.

6.6 Geração de Colunas

Diferente dos resultados para o PEU mostrados no trabalho de Vance [57], o número de nós com restrições de aresta é proporcionalmente grande em relação ao número de nós sem estas restrições, para o PEBF2. Este fato encarece demasiadamente o custo do algoritmo em relação ao tempo gasto na geração de colunas.

Duas abordagens são possíveis: a geração de colunas através do resolvedor, ou, o teste de todas combinações possíveis dos itens envolvidos nas restrições de aresta. A Tabela 6.7 mostra a comparação entre as duas abordagens. As colunas `Nome` e `Itens` referem-se ao nome e quantidade de itens da instância. As demais seções da tabela são divididas em duas colunas: `Res.` indica que a geração de colunas com restrições de aresta foi feita pelo resolvedor de programação linear inteira; e a coluna `Comb.` indica que foram testadas todas combinações possíveis dos itens que compõem as restrições de aresta. A seção `Obj.` indica o valor da solução encontrada e a seção `Opt?` se esses valores são ótimos. As seções `Variáveis` e `Nós` indicam a quantidade de variáveis com custo reduzido negativo geradas e nós da árvore gerados. A seção `Geração de Colunas(s)` indica o tempo total gasto na geração das colunas e a seção `Total(s)` o tempo total do algoritmo, em segundos de CPU. O tempo máximo de execução foi limitado à 1200 segundos de CPU.

A geração de colunas pelas combinações foi mais rápida em 12 dos 39 casos testados e, mesmo assim, para instâncias cujo número de variáveis geradas foi relativamente pequeno. A diferença de tempo de execução destas instâncias foi muito pequena, em média 0,09 segundos. A única exceção foi a instância `ht_c3_01`, onde gerar todas combinações foi muito rápido (9,26 segundos contra 79,09 segundos do resolvedor). Podemos notar que, de forma geral, o número de variáveis (com custo reduzido negativo) geradas foi muito maior pelo resolvedor que pelo exame das combinações devido, principalmente, à complexidade que as restrições de aresta tomam durante a execução do algoritmo. Outro fato importante a ser considerado é que a geração das colunas pelo exame das combinações analisa todas alturas possíveis, para cada combinação. Quando geramos colunas pelo resolvedor, a primeira destas que apresentar

Tabela 6.7: Comparação da geração de colunas pelo resolvidor e geração por todas combinações.

Nome	Itens	Obj.		Opt?		Variáveis		Nós		Geração de colunas (s)		Total (s)	
		Res.	Comb.	Res.	Comb.	Res.	Comb.	Res.	Comb.	Res.	Comb.	Res.	Comb.
beng01	20	36	36	*	*	15554	14055	4039	4533	115.71	434.37	141.74	491.60
beng02	40	61	61	*	*	6525	4070	933	1167	128.08	1098.64	153.29	1202.55
beng03	60	88	88			34319	1337	8793	437	1007.04	1180.74	1200.02	1205.21
beng04	80	113	113			19108	1015	4599	269	997.69	1265.41	1200.24	1291.68
beng05	100	138	138			16912	874	3929	191	946.66	1438.47	1202.32	1473.78
beng06	40	40	40			28356	2394	4576	452	999.73	1198.32	1200.30	1203.34
beng07	80	70	60			15571	1523	1931	391	951.79	1253.56	1200.27	1259.56
beng08	120	105	105			13045	1281	1431	324	799.89	1211.74	1200.12	1220.18
beng09	160	129	129			11102	1309	1200	263	683.54	1103.97	1200.97	1206.32
beng10	200	159	159			7003	947	589	131	422.02	1270.30	1200.29	1275.62
cgcut1	7	14	14	*	*	10	10	5	5	0.01	0	0.02	0.02
cgcut2	10	51	51	*	*	144	139	29	29	0.57	0.06	0.73	0.21
cgcut3	20	229	229			44533	8916	28137	6859	1105.59	1181.01	1200.01	1200.04
gcut01	10	1016	1016	*	*	0	0	1	1	0	0	0	0
gcut02	20	1262	1262	*	*	44	44	9	9	0.08	0.05	0.13	0.10
gcut03	30	1810	1810	*	*	37	37	1	1	0.07	0.07	0.11	0.10
gcut04	50	3126	3126			6786	1950	3251	1285	1175.59	1193.29	1200.79	1201.03
gcut05	10	1360	1360	*	*	14	14	1	1	0.01	0.01	0.02	0.02
gcut06	20	2862	2862	*	*	309	303	209	197	6.13	20.02	6.78	20.64
gcut07	30	4979	4979	*	*	1579	1154	1201	1029	88.37	1197.35	92.78	1200.90
gcut08	50	6168	6168	*	*	104	107	3	3	1.10	1.13	1.28	1.31
gcut09	10	2646	2646	*	*	31	30	31	31	0.22	0.15	0.28	0.21
gcut10	20	6167	6167	*	*	22	22	1	1	0.08	0.08	0.10	0.09
gcut11	30	7298	7298	*	*	43	43	1	1	0.33	0.33	0.38	0.38
gcut12	50	14943	14943	*	*	40	40	1	1	0.82	0.81	0.86	0.87
gcut13	32	5398	5415			17543	1012	1741	183	1132.14	1589.32	1200.14	1593.05
ht_c1_01	16	27	27	*	*	50	45	3	3	0.08	0.01	0.12	0.05
ht_c1_02	17	29	29	*	*	51	51	1	1	0.01	0.01	0.06	0.06
ht_c1_03	16	23	23	*	*	76	76	1	1	0.01	0.01	0.10	0.09
ht_c2_01	25	20	20	*	*	203	203	1	1	0.05	0.05	0.80	0.82
ht_c2_02	25	34	34	*	*	256	245	3	3	0.12	0.08	1.17	1.07
ht_c2_03	25	23	23	*	*	220	220	1	1	0.05	0.06	0.91	0.90
ht_c3_01	28	40	40	*	*	4222	1836	83	75	60.19	3.27	79.09	9.26
ht_c3_02	29	42	42	*	*	424	360	3	3	1.73	0.26	4.13	2.06
ht_c3_03	28	43	43	*	*	515	340	5	5	1.67	0.19	4.36	1.73
ht_c5_01	73	101	101			9342	2520	517	303	925.09	1242.68	1201.17	1323.63
ht_c5_02	73	106	106			8329	3495	273	251	744.62	1035.47	1229.37	1235.54
ht_c5_03	73	106	106			9956	1566	719	151	884.43	1519.50	1200.36	1560.17
ht_c6_01	97	136	136			5647	3380	313	327	834.41	1000.65	1206.92	1205.46
ht_c6_02	97	145	145			5723	3261	245	235	657.95	878.94	1233.19	1204.75
ht_c6_03	97	139	139			6764	2527	125	149	781.73	1160.98	1225.09	1341.61
ht_c7_01	196	261	261			2365	2364	1	1	63.26	63.30	1200.82	1200.61
ht_c7_02	197	283	283			2194	2195	1	1	93.73	93.81	1200.77	1201.20
ht_c7_03	196	273	273			2363	2365	1	1	87.84	87.33	1200.36	1201.31

custo reduzido negativo é devolvida. Em testes posteriores deste capítulo, são mostrados resultados considerando sempre a primeira coluna com custo reduzido negativo.

Desta maneira, o processo de ramificação foi modificado com o intuito de gerar uma estrutura comportada das restrições de aresta. Na ramificação simples, utilizada até agora, não há restrições quanto a escolha dos itens para a ramificação. A segunda abordagem escolhe itens que ainda não foram considerados em ramificações anteriores (vide Seção 5.2.1 para mais detalhes). A Tabela 6.8 mostra o efeito da ramificação simples sobre algumas instâncias e a Tabela 6.9 o efeito da ramificação que não permite itens já considerados em ramificações anteriores. Sua estrutura é idêntica à Tabela 6.7.

Podemos notar que a segunda abordagem foi mais lenta na maioria dos casos testados, tanto para o resolvidor como para o teste das combinações (embora esse resultado já fosse esperado para este último). Optamos pela ramificação simples.

Uma estratégia para acelerar o processo, foi testar um determinado número de combinações dos itens que compõem as restrições de aresta e, caso não encontremos uma

Tabela 6.8: Comparação da geração de colunas pelo resolvidor e geração por todas combinações utilizando ramificação simples.

Nome	Itens	Obj		Opt?		Variáveis		Nós		Geração de colunas (s)		Total (s)	
		Res.	Comb	Res.	Comb	Res.	Comb	Res.	Comb	Res.	Comb	Res.	Comb
ht_c1_01	16	27.00	27.00	*	*	20	15	1	1	0.23	0.00	0.37	0.15
ht_c1_02	17	29.00	29.00	*	*	0	0	0	0	0.00	0.00	0.16	0.13
ht_c1_03	16	23.00	23.00	*	*	0	0	0	0	0.00	0.00	0.23	0.24
ht_c2_01	25	20.00	20.00	*	*	0	0	0	0	0.00	0.00	1.76	1.83
ht_c2_02	25	34.00	34.00	*	*	14	3	1	1	0.08	0.01	2.56	2.39
ht_c2_03	25	23.00	23.00	*	*	0	0	0	0	0.00	0.00	2.02	2.06
ht_c3_01	28	40.00	40.00	*	*	4119	1278	76	52	155.02	6.87	197.56	18.65
ht_c3_02	29	42.00	42.00	*	*	94	30	1	1	4.07	0.09	9.36	4.37
ht_c3_03	28	43.00	43.00	*	*	240	65	2	2	4.32	0.14	10.13	3.69
ht_c4_01	49	74.00	74.00			8626	4742	728	814	1033.26	1116.72	1201.28	1213.44
ht_c4_02	49	74.00	74.00			8024	2471	208	212	1024.62	1155.95	1201.46	1211.61
ht_c4_03	49	80.00	80.00			7722	4516	343	457	1007.77	1100.70	1217.83	1204.74

Tabela 6.9: Comparação da geração de colunas pelo resolvidor e geração por todas combinações utilizando ramificação com itens diferentes.

Nome	Itens	Obj		Opt?		Variáveis		Nós		Geração de colunas (s)		Total (s)	
		Res.	Comb	Res.	Comb	Res.	Comb	Res.	Comb	Res.	Comb	Res.	Comb
ht_c1_01	16	27.00	27.00	*	*	20	15	1	1	0.17	0.01	0.26	0.10
ht_c1_02	17	29.00	29.00	*	*	0	0	0	0	0.00	0.00	0.11	0.11
ht_c1_03	16	23.00	23.00	*	*	0	0	0	0	0.00	0.00	0.18	0.17
ht_c2_01	25	20.00	20.00	*	*	0	0	0	0	0.00	0.00	1.24	1.27
ht_c2_02	25	34.00	34.00	*	*	14	3	1	1	0.05	0.00	1.79	1.66
ht_c2_03	25	23.00	23.00	*	*	0	0	0	0	0.00	0.00	1.39	1.40
ht_c3_01	28	40.00	40.00	*	*	3626	1861	74	90	69.08	7.02	96.87	18.35
ht_c3_02	29	42.00	42.00	*	*	94	30	1	1	2.69	0.06	6.55	3.12
ht_c3_03	28	43.00	43.00	*	*	449	125	5	5	7.47	0.37	13.28	3.32
ht_c4_01	49	74.00	74.00			14093	2714	1394	480	1007.10	1190.23	1200.86	1230.78
ht_c4_02	49	74.00	74.00			13270	2911	390	270	1017.63	1203.17	1201.32	1244.78

variável com o custo reduzido negativo, utilizar o resolvidor para tal (Seção 5.2.2). As Tabelas 6.10 e 6.11 mostram os resultados obtidos com o teste de 5, 6 e 8 itens, respectivamente. Estes itens são escolhidos aleatoriamente dos itens que compõem as restrições de aresta. As instâncias testadas foram escolhidas devido seu comportamento nos ensaios anteriores. São 8 instâncias, 4 com solução ótima obtida e 4 sem solução ótima em 1200 segundos de CPU. Os resultados destas tabelas foram obtidos na máquina de apoio descrita na Seção 6.1.

A Tabela 6.10 contém os campos: Nome e Itens, como nas demais tabelas; Opt?, indicando se uma solução ótima foi encontrada; Nós, indicando o número de nós com restrições de aresta; Vars . PLI, indicando o número de variáveis, com custo reduzido negativo, geradas pelo resolvidor; Vars . Comb., idem anterior mas variáveis geradas pelo teste das combinações; e Total de Vars., indicando o número total de variáveis geradas. O valores da solução obtidos para cada instância, foi igual em todos os casos (5, 6 e 8 itens).

A Tabela 6.11 contém os campos: Nome e Itens, como nas demais tabelas; Ger . PLI, indicando o tempo gasto na geração de colunas através do resolvidor; Ger . Comb., indicando o tempo gasto na geração de colunas pela análise das combinações; Ger . Total, indicando o tempo total na geração de colunas; e Total, indicando o tempo total gasto pelo algoritmo. Todos os tempos estão em segundos de CPU.

Tabela 6.10: Comparação entre números de itens escolhidos das restrições de aresta: quantidade de variáveis.

Nome	Itens	Opt?			Nós			Vars. PLI			Vars. Comb.			Total de Vars.		
		5	6	8	5	6	8	5	6	8	5	6	8	5	6	8
beng01	20	*	*	*	3947	4513	4019	6254	6850	4085	7355	9771	9430	13672	16684	13578
beng02	40	*	*	*	875	1407	577	1392	2811	452	2746	4169	2024	4275	7117	2613
beng03	60				7422	6150	5196	11485	9630	7924	15521	14618	11786	27295	24537	19710
cgcut3	20			*	18329	17721	11733	16399	14655	13160	10668	11735	12030	27111	26434	25190
gcut07	30	*	*	*	1114	1090	1096	655	561	490	736	772	869	1421	1363	1389
ht_c3_01	28	*	*	*	62	62	62	0	0	0	1581	1581	1581	1831	1831	1831
ht_c4_01	49				2188	2054	1677	5590	4647	3397	12480	11815	9330	18560	16952	12727
ht_c5_01	73				1190	1378	914	5126	3521	2795	8845	9518	5785	14970	14038	8580

Tabela 6.11: Comparação entre números de itens escolhidos das restrições de aresta: tempos.

Nome	Itens	Ger. PLI			Ger. Comb.			Ger. Total			Total		
		5	6	8	5	6	8	5	6	8	5	6	8
beng01	20	151.47	169.70	115.37	20.98	38.84	67.54	173.43	209.67	183.91	218.97	262.87	229.43
beng02	40	80.85	152.11	36.04	16.27	45.82	30.89	97.59	198.72	67.22	123.73	243.46	83.47
beng03	60	812.83	751.83	549.24	161.47	246.56	496.34	978.09	1001.85	1048.42	1200.36	1200.36	1201.76
cgcut3	20	1019.23	977.15	911.18	71.32	116.20	172.14	1092.55	1095.46	1085.30	1200.01	1200.03	1257.44
gcut07	30	124.20	118.68	104.20	88.33	138.01	273.87	212.84	257.00	378.38	220.95	264.89	386.31
ht_c3_01	28	0.00	0.00	0.00	4.91	4.87	4.89	5.19	5.19	5.18	15.13	15.09	15.02
ht_c4_01	49	679.97	576.61	427.81	294.44	413.99	613.78	977.38	993.32	1043.46	1205.34	1200.67	1207.24
ht_c5_01	73	608.81	459.77	373.59	266.68	415.43	628.19	879.56	879.12	1003.86	1201.28	1200.26	1210.05

A escolha de 5, 6 e 8 itens levam, respectivamente, a 32, 64 e 256 possíveis combinações a serem testadas. Destas, somente aquelas que formarem conjuntos independentes com relação ao grafo formado pelas restrições de aresta, são consideradas (vide Seção 5.2.2). Podemos notar que a variação na quantidade de itens não influenciou muito nos tempos de resolução. No geral, o teste de 5 itens foi mais eficiente. Um dos fatores que contribuiu para isso, foi o baixo número de combinações inutilmente testadas, em nós cujo este tipo de teste foi ineficaz, ou seja, o teste das combinações não resultou em nenhuma variável com custo reduzido negativo e, portanto, o resolvidor teve que ser utilizado para gerar uma coluna.

6.7 Resultados Finais

As Tabelas 6.12 e 6.13 mostram os melhores resultados obtidos. Nelas utilizamos o teste de combinações de no máximo 5 itens que porventura venham a compor restrições de aresta. Caso este teste falhe, é utilizado o resolvidor. Em ambos casos, a primeira variável com custo reduzido negativo é devolvida. Foram utilizados os algoritmos da mochila gulosa para geração do limitante superior inicial e o BFDH para geração dos demais limitantes superiores.

A Tabela 6.12 segue a descrição da Tabela 6.5. Uma solução ótima foi encontrada em 24 instâncias (51,06%). O *gap* médio obtido foi de 1,38%, com desvio padrão 1,72. Entre as 23 instâncias (48,94%) onde nenhuma solução ótima foi encontrada, o *gap* médio foi de 2,88%, com desvio padrão 1,40. Estes resultados são ligeiramente piores

que a configuração do algoritmo na Seção 6.5, mas o tempo de execução nas instâncias onde o ótimo foi encontrado, é melhor, como pode ser visto na Tabela 6.14.

A descrição da Tabela 6.13 é idêntica a Tabela 6.6 mas as seções Variáveis e Geração de Colunas contém mais uma coluna: Comb., que indica o número de variáveis e o tempo gasto na geração de colunas pelo teste das combinações.

A Tabela 6.14 mostra a comparação entre duas configurações do algoritmo: a coluna Básica considera a configuração utilizando apenas o resolvidor para geração de colunas em nós com restrições de aresta; a coluna PCRN (Primeiro Custo Reduzido Negativo) considera a configuração descrita no começo desta seção. A seção Gap indica o *gap* proporcional entre o melhor limitante inferior e o valor da melhor solução encontrada. As seções Opt? e Tempo(s) da tabela indicam se uma solução ótima foi encontrada, e o tempo gasto para isso, em segundos de CPU, respectivamente. A coluna Dif.(s) mostra a diferença, em segundos de CPU, do tempo gasto nas duas configurações. A coluna Ganho mostra o ganho de velocidade entre as configurações, que é calculado através da equação

$$Ganho = \frac{tempo(Básica) - tempo(PCRN)}{tempo(PCRN)}.$$

Note que quando o ganho é positivo, a configuração PCRN é mais rápida que a configuração básica. Quando negativo, a situação se inverte.

O algoritmo com a configuração PCRN foi mais rápido em 11 (45,83%) dos 24 casos onde ambas configurações alcançaram uma solução ótima. Destes, 8 casos (33,33%) foram executados mais rápido pela configuração básica e 5 (20,83%) tiveram o mesmo tempo de execução.

No que se refere a configuração PCRN, destacam-se as instâncias do grupo ht_c3, com mais de 2 vezes de ganho. Em especial, a instância ht_c3_01 (marcada com * na tabela) teve o excepcional ganho de mais de 16 vezes e uma diferença de mais de 2 minutos de execução. Outro caso interessante foi a resolução da instância beng01 (marcada com † na tabela) onde a diferença de tempo de execução foi de mais de 13 minutos de execução. Em verdade, a média de ganhos da configuração PCRN, excluindo o caso extremo da instância ht_c3_01, foi de 1,82 e desvio padrão de 1,68.

No que se refere a configuração básica, o ganho não passou de 0,13 em favor desta, com desvio de 0,10. A diferença de tempo de execução também foi muito pequena, raramente passando de 1 segundo, com exceção da instância gcut07 (marcada com ‡) onde a diferença foi de mais de 1 minuto. Mas estes resultados não são favoráveis à configuração básica, já que em 6 instâncias, das 8 onde a configuração básica foi mais rápida, somente foi necessária a geração de colunas na fase inicial e nenhuma ramificação foi feita. Ou seja, foram gerados o mesmo número de variáveis, e a diferença de tempo, que é muito pequena, pode ser causada por variações na utilização da CPU.

Tabela 6.12: Resultados considerando a primeira variável com custo reduzido negativo: características da árvore.

Nome	Itens	Limitantes			Obj.	Dif.	Gap	Opt?	Nós			
		P. Inf.	M. Inf.	Sup.					Liv.	Res.	Total	Prof.
beng01	20	33	36	36	36	0	0.00	*	4	3811	3815	27
beng02	40	60	61	61	61	0	0.00	*	11	3698	3709	23
beng03	60	86	86	88	88	2	2.33		18	16013	16031	34
beng04	80	108	108	113	112	4	3.70		39	10626	10665	38
beng05	100	134	134	138	138	4	2.99		36	8641	8677	36
beng06	40	37	38	40	40	2	5.26		22	15469	15491	28
beng07	80	68	68	71	71	3	4.41		49	7356	7405	48
beng08	120	101	101	105	105	4	3.96		75	4412	4487	74
beng09	160	126	126	129	129	3	2.38		83	3366	3449	91
beng10	200	156	156	159	159	3	1.92		9	1728	1737	123
cgcut1	7	12	14	14	14	0	0.00	*	3	2	5	2
cgcut2	10	46	51	51	51	0	0.00	*	4	39	43	7
cgcut3	20	220	228	230	229	1	0.44		8	39347	39355	32
gcut01	10	1016	1016	1016	1016	0	0.00	*	1	0	1	0
gcut02	20	1262	1262	1262	1262	0	0.00	*	8	17	25	7
gcut03	30	1810	1810	1810	1810	0	0.00	*	1	0	1	0
gcut04	50	3108	3124	3159	3126	2	0.06		4	5149	5153	27
gcut05	10	1360	1360	1360	1360	0	0.00	*	1	0	1	0
gcut06	20	2792	2862	2862	2862	0	0.00	*	7	126	133	10
gcut07	30	4894	4979	4979	4979	0	0.00	*	4	1127	1131	18
gcut08	50	6149	6168	6168	6168	0	0.00	*	20	439	459	19
gcut09	10	2509	2646	2646	2646	0	0.00	*	4	25	29	6
gcut10	20	6167	6167	6167	6167	0	0.00	*	1	0	1	0
gcut11	30	7298	7298	7298	7298	0	0.00	*	1	0	1	0
gcut12	50	14943	14943	14943	14943	0	0.00	*	1	0	1	0
gcut13	32	5090	5103	5415	5398	295	5.78		20	2527	2547	26
ht_c1_01	16	26	27	27	27	0	0.00	*	2	1	3	1
ht_c1_02	17	29	29	29	29	0	0.00	*	1	0	1	0
ht_c1_03	16	23	23	23	23	0	0.00	*	1	0	1	0
ht_c2_01	25	20	20	20	20	0	0.00	*	1	0	1	0
ht_c2_02	25	33	34	34	34	0	0.00	*	3	2	5	2
ht_c2_03	25	23	23	23	23	0	0.00	*	1	0	1	0
ht_c3_01	28	37	40	40	40	0	0.00	*	10	13	23	9
ht_c3_02	29	41	42	42	42	0	0.00	*	2	1	3	1
ht_c3_03	28	42	43	43	43	0	0.00	*	4	3	7	3
ht_c4_01	49	71	73	74	74	1	1.37		15	3060	3075	35
ht_c4_02	49	73	73	74	74	1	1.37		19	3436	3455	41
ht_c4_03	49	72	73	75	75	2	2.74		13	2997	3010	33
ht_c5_01	73	103	104	107	107	3	2.88		27	1962	1989	28
ht_c5_02	73	104	104	106	106	2	1.92		29	1851	1880	29
ht_c5_03	73	103	103	107	106	3	2.91		32	1749	1781	31
ht_c6_01	97	131	132	136	136	4	3.03		35	1212	1247	34
ht_c6_02	97	139	140	145	145	5	3.57		22	627	649	38
ht_c6_03	97	137	137	139	139	2	1.46		27	848	875	36
ht_c7_01	196	254	254	261	261	7	2.76		99	180	279	98
ht_c7_02	197	272	272	283	282	10	3.68		71	122	193	70
ht_c7_03	196	263	263	273	273	10	3.80		110	137	247	103

Tabela 6.13: Resultados considerando a primeira variável com custo reduzido negativo: geração de colunas.

Nome	Itens	Variáveis				Geração de colunas (s)				Tempo (s)		
		Moc.	PLI	Comb.	Total	Moc.	PLI	Comb.	Total	Heur.	Ram.	Total
beng01	20	46	8961	4687	13694	0.01	158.35	16.22	175.65	0.41	2.52	220.70
beng02	40	120	19353	12416	31889	0.04	624.37	48.27	675.88	0.71	4.40	841.94
beng03	60	237	49673	26933	76843	0.09	2783.30	132.91	2927.47	2.18	26.42	3600.96
beng04	80	427	33162	14256	47845	0.27	2751.87	90.19	2851.58	2.07	23.54	3600.26
beng05	100	479	29694	11747	41920	0.33	2754.51	66.08	2830.27	1.86	21.84	3600.82
beng06	40	351	63782	47775	111908	0.16	2624.44	165.94	2804.54	1.48	23.45	3600.53
beng07	80	723	34300	17237	52260	0.56	2629.38	60.80	2700.94	1.14	19.65	3600.19
beng08	120	1072	29687	13682	44441	1.21	2008.24	71.60	2093.58	1.01	19.66	3600.07
beng09	160	1553	23361	10843	35757	2.33	1628.99	75.05	1720.96	0.92	24.94	3600.52
beng10	200	1221	16477	4146	21844	2.50	1145.50	53.16	1212.29	0.61	19.18	3604.71
cgcut1	7	11	0	0	11	0.00	0.00	0.00	0.00	0.00	0.00	0.03
cgcut2	10	26	2	176	204	0.01	0.08	0.07	0.17	0.00	0.02	0.57
cgcut3	20	53	56255	16276	72584	0.03	3184.70	107.42	3298.09	3.47	25.51	3600.06
gcut01	10	0	0	0	0	0.00	0.00	0.00	0.00	0.00	0.00	0.01
gcut02	20	34	5	22	61	0.05	0.41	0.11	0.57	0.00	0.01	0.75
gcut03	30	2	0	0	2	0.01	0.00	0.00	0.01	0.00	0.00	0.02
gcut04	50	96	5878	7207	13181	0.88	2659.45	846.62	3508.25	0.74	6.00	3600.24
gcut05	10	8	0	0	8	0.01	0.00	0.00	0.01	0.00	0.00	0.02
gcut06	20	33	89	125	247	0.10	6.70	3.40	10.21	0.01	0.08	11.09
gcut07	30	29	707	768	1504	0.19	135.56	84.07	219.96	0.11	0.77	228.80
gcut08	50	144	860	1141	2145	2.27	405.51	174.02	581.99	0.08	0.54	595.40
gcut09	10	12	3	13	28	0.02	0.20	0.19	0.41	0.00	0.01	0.52
gcut10	20	14	0	0	14	0.09	0.00	0.00	0.09	0.00	0.00	0.12
gcut11	30	42	0	0	42	0.58	0.00	0.00	0.58	0.00	0.00	0.68
gcut12	50	24	0	0	24	0.90	0.00	0.00	0.90	0.00	0.00	0.97
gcut13	32	328	15670	6078	22076	8.30	2166.86	1292.53	3470.14	0.29	3.66	3603.82
ht_c1_01	16	31	0	8	39	0.00	0.00	0.00	0.00	0.00	0.00	0.08
ht_c1_02	17	46	0	0	46	0.01	0.00	0.00	0.01	0.00	0.00	0.10
ht_c1_03	16	94	0	0	94	0.02	0.00	0.00	0.02	0.00	0.00	0.22
ht_c2_01	25	333	0	0	333	0.14	0.00	0.00	0.16	0.00	0.00	2.61
ht_c2_02	25	271	0	24	295	0.17	0.00	0.02	0.22	0.00	0.01	2.12
ht_c2_03	25	223	0	0	223	0.10	0.00	0.00	0.12	0.00	0.00	1.47
ht_c3_01	28	250	59	448	757	0.15	1.75	0.35	2.33	0.00	0.03	7.69
ht_c3_02	29	297	0	35	332	0.38	0.00	0.05	0.47	0.00	0.00	2.90
ht_c3_03	28	280	0	177	457	0.25	0.00	0.17	0.45	0.00	0.01	3.58
ht_c4_01	49	449	32837	15712	48998	0.68	2946.62	150.19	3104.38	0.55	6.36	3604.65
ht_c4_02	49	387	33785	17934	52106	0.52	3002.46	152.49	3155.44	0.49	6.92	3610.47
ht_c4_03	49	401	29531	13720	43652	0.49	2947.68	149.21	3097.38	0.63	7.24	3607.12
ht_c5_01	73	801	20453	5947	27201	1.78	2899.32	53.95	2955.05	0.52	7.31	3610.85
ht_c5_02	73	734	21461	4308	26503	1.73	2943.45	51.72	2996.90	0.41	5.95	3607.76
ht_c5_03	73	875	19117	5740	25732	1.85	2840.13	54.06	2900.56	0.33	6.12	3605.29
ht_c6_01	97	1319	17226	7901	26446	6.08	2529.41	80.97	2622.54	0.26	7.08	3613.85
ht_c6_02	97	1697	8927	3273	13897	11.90	2593.24	68.04	2676.63	0.16	5.28	3638.46
ht_c6_03	97	1591	11308	4217	17116	9.23	2671.12	77.43	2757.78	0.31	6.01	3603.21
ht_c7_01	196	4487	997	613	6097	105.14	2516.85	38.07	2662.64	0.11	90.98	3751.82
ht_c7_02	197	5970	844	1030	7844	227.44	826.15	66.24	1124.05	0.09	201.08	3713.49
ht_c7_03	196	5747	1023	1570	8340	123.12	2891.35	70.13	3084.6	0.15	97.27	3727.31

Tabela 6.14: Comparação entre configurações básica e PCRN do algoritmo.

Nome	Itens	Gap		Opt?		Tempo (s)		Dif. (s)	Ganho
		Básica	PCRN	Básica	PCRN	Básica	PCRN		
beng01	20			*	*	282.70	220.70	62.00	0.28
beng02	40			*	*	1661.04	841.94	819.10	0.97 †
beng03	60	2.33	2.33			3600.03	3600.96		
beng04	80	3.70	3.70			3600.43	3600.26		
beng05	100	2.99	2.99			3600.33	3600.82		
beng06	40	5.26	5.26			3600.90	3600.53		
beng07	80	2.94	4.41			3600.80	3600.19		
beng08	120	3.96	3.96			3600.36	3600.07		
beng09	160	2.38	2.38			3602.92	3600.52		
beng10	200	1.92	1.92			3600.88	3604.71		
cgcut1	7			*	*	0.03	0.03	0.00	0.00
cgcut2	10			*	*	1.95	0.57	1.38	2.42
cgcut3	20	0.44	0.44			3600.03	3600.06		
gcut01	10			*	*	0.01	0.01	0.00	0.00
gcut02	20			*	*	0.57	0.75	0.18	-0.24
gcut03	30			*	*	0.02	0.02	0.00	0.00
gcut04	50	0.10	0.06			3600.71	3600.24		
gcut05	10			*	*	0.03	0.02	0.01	0.50
gcut06	20			*	*	9.97	11.09	1.12	-0.10
gcut07	30			*	*	163.13	228.80	65.67	-0.29 ‡
gcut08	50			*	*	655.45	595.40	60.05	0.10
gcut09	10			*	*	0.74	0.52	0.22	0.42
gcut10	20			*	*	0.11	0.12	0.01	-0.08
gcut11	30			*	*	0.60	0.68	0.08	-0.12
gcut12	50			*	*	0.83	0.97	0.14	-0.14
gcut13	32	5.78	5.78			3601.06	3603.82		
ht_c1_01	16			*	*	0.28	0.08	0.20	2.50
ht_c1_02	17			*	*	0.10	0.10	0.00	0.00
ht_c1_03	16			*	*	0.22	0.22	0.00	0.00
ht_c2_01	25			*	*	2.57	2.61	0.04	-0.02
ht_c2_02	25			*	*	12.96	2.12	10.84	5.11 ◇
ht_c2_03	25			*	*	1.43	1.47	0.04	-0.03
ht_c3_01	28			*	*	136.15	7.69	128.46	16.70 *
ht_c3_02	29			*	*	9.01	2.90	6.11	2.11 ◇
ht_c3_03	28			*	*	17.06	3.58	13.48	3.77 ◇
ht_c4_01	49	1.37	1.37			3607.95	3604.65		
ht_c4_02	49	1.37	1.37			3601.66	3610.47		
ht_c4_03	49	2.74	2.74			3605.31	3607.12		
ht_c5_01	73	2.88	2.88			3607.37	3610.85		
ht_c5_02	73	1.92	1.92			3608.42	3607.76		
ht_c5_03	73	2.91	2.91			3611.48	3605.29		
ht_c6_01	97	3.03	3.03			3607.67	3613.85		
ht_c6_02	97	3.57	3.57			3615.72	3638.46		
ht_c6_03	97	1.46	1.46			3601.49	3603.21		
ht_c7_01	196	2.76	2.76			3703.50	3751.82		
ht_c7_02	197	3.68	3.68			3626.62	3713.49		
ht_c7_03	196	3.80	3.80			5385.75	3727.31		

O ganho de velocidade da configuração PCRN se deve, principalmente, pelo grande número de variáveis com custo reduzido negativo geradas pelo teste de combinações. Em 3 casos (marcados com \diamond), não houve a necessidade de se utilizar o resolvidor para gerar variáveis, como pode ser visto na Tabela 6.13.

Nas instâncias onde não foi provada a otimalidade, os *gaps* permaneceram os mesmos entre as configurações, com apenas duas exceções: a instância *beng07*, onde a configuração PCRN apresentou um *gap* de 4,41, pior que o *gap* da configuração básica de 2,94. Em contrapartida, para a instância *gcut04*, a configuração PCRN teve o *gap* de 0,06 e a configuração básica o *gap* de 0,10.

Podemos concluir que a configuração PCRN é a melhor escolha, devido a seu desempenho na geração de colunas.

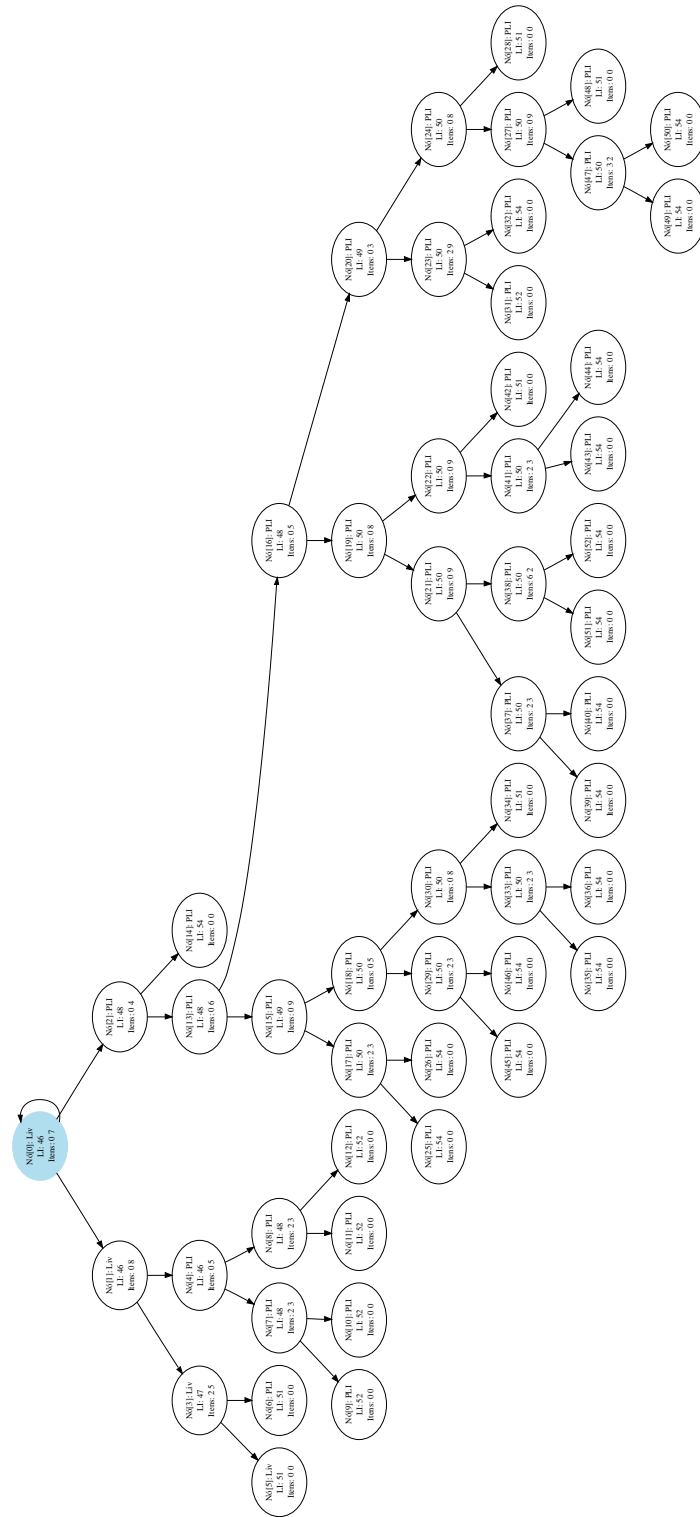


Figura 6.3: Árvore de ramificação da instância cgcut2.

Capítulo 7

Conclusão e Considerações Finais

Neste trabalho concentramos nossos esforços para resolução do problema de empacotamento bidimensional em faixas com 2 estágios PEBF2, já que é um importante problema na indústria. Apresentamos alguns algoritmos aproximados e uma heurística para resolução do mesmo, e um algoritmo exato que se utiliza de pequenas variações dos algoritmos aproximados apresentados, para cálculo de limitantes primais. Apresentamos também métodos de geração de colunas e analisamos sua sensibilidade quanto às modificações do subproblema.

Os algoritmos utilizados para cálculo dos limitantes superiores apresentaram bom desempenho, com destaque para o algoritmo da mochila gulosa, cujo *gap* entre o valor da solução ótima e o valor da solução encontrada por este foi pequeno (em média, 2,01%). Além disso, o algoritmo da mochila gulosa é um algoritmo rápido e é um excelente candidato para ser utilizado em casos reais onde o tamanho dos itens e a largura da faixa são numericamente grandes, devido sua capacidade de aproveitar a área de cada item como informação importante no processo de empacotamento. Junto dele, o BFDH apresentou bons resultados também, mas em instâncias numericamente menores.

O limitante inferior apresentou uma qualidade extraordinária. O *gap* médio foi de 1,38%, com desvio padrão de 1,72, o que atesta a qualidade da formulação utilizada.

A geração de colunas em nós livre de restrições de aresta foi muito rápida, devido ao bom comportamento do algoritmo de programação dinâmica da mochila. Em contrapartida, a geração de colunas em nós com restrições de aresta foi lenta. A principal razão foi a complexidade que estas restrições tomam durante as ramificações.

O comportamento do algoritmo foi muito diferente do esperado. Como pudemos observar no Capítulo 6, as árvores resultantes são bem balanceadas, o que gerou um grande número de nós com restrições de aresta. Esta desproporção acarretou sérias consequências na geração de colunas, que foi majoritariamente feita, ou pelo resolve-

dor, ou pelo teste das combinações válidas dos itens componentes das restrições de aresta. Esperávamos árvores desbalanceadas, como no caso do problema de empacotamento unidimensional PEU (vide Seção 6.3), o que facilitaria a geração de colunas. Um das razões para este comportamento é o *gap* para problemas unidimensionais é muito pequeno, e um dos indícios deste fato é a conjectura MIRUP (vide Seção 3.3). Esta conjectura aparentemente não é válida para problema bidimensionais.

Portanto, o maior tempo gasto na execução do algoritmo foi na geração de colunas em nós com restrições de aresta, principalmente quando tratada pelo resolvidor. Mesmo nas instâncias onde o teste das combinações gerou mais colunas (com custo reduzido negativo) que o resolvidor, o tempo total gasto por este último ainda foi maior. Infelizmente, como mostrado na Seção 6.6, o teste de todas combinações se torna proibitivo. A estratégia de analisar apenas algumas combinações, e caso necessário, utilizar o resolvidor, sempre retornando a primeira coluna com custo reduzido negativo, não apresentou ganhos expressivos, principalmente no que consta à prova de otimalidade.

Concluimos que, embora o algoritmo de *branch-and-price* seja promissor para resolução do PEBF2, devido à qualidade dos limitantes, é preciso mais estudos e avanços na geração de colunas envolvendo restrições de aresta para que se acelere o processo e o algoritmo se torne viável para instâncias de grande porte.

Um bom indicativo para trabalhos futuros seria aprofundar os estudos sobre as restrições de aresta, já que são componentes de problemas como problemas de conjuntos independentes e partição em cliques. O desenvolvimento de heurísticas para geração dessas colunas também deve ser considerado.

Uma outra alternativa seria utilizar uma formulação baseada em fluxos, onde os itens a serem empacotados estão relacionados entre si por um grafo onde cada nó representa um possível tamanho de um nível. Fekete e Schepers [23, 24, 25] propuseram esse modelo. Carvalho [13, 14] apresentam formulações deste tipo para problema de corte e estoque unidimensional e Belov *et al.* [4] apresentaram um algoritmo de *branch-and-cut-and-price* para o mesmo problema.

Referências Bibliográficas

- [1] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46(3):316–329, 1998.
- [2] M. S. Bazaraa, J. J. Jarvis, and H. F. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, New York, NY, USA, second edition, 1990.
- [3] J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.
- [4] G. Belov, A. Letchford, and E. Uchoa. A Node-Flow Model for One-Dimensional Stock Cutting: Robust Branch & Cut & Price. Technical Report 7, Universidade Federal Fluminense, 2005.
- [5] B. Bengtsson. Packing rectangular pieces - A heuristic approach. *The Computing Journal*, 25:353–357, 1982.
- [6] G. Brassard and P. Bratley. *Fundamentals of algorithmics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.
- [7] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25:30–44, 1977.
- [8] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.
- [9] E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal of Computing*, 9:801–826, 1980.
- [10] COIN. COmputational INfrastructure for Operations Research.
<http://www.coin-or.org>.
- [11] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

- [12] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.
- [13] J. M. V. de Carvalho. Exact Solution of Cutting Stock Problems Using Column Generation and Branch-and-Bound. *Int. Trans. Opl. Res.*, 5(1):35–44, 1998.
- [14] J. M. V. de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141:253–273, September 2002.
- [15] Z. Degraeve and L. Schrage. Optimal Integer Solutions to Industrial Cutting Stock Problems. *INFORMS Journal on Computing*, 11:406–419, 1999.
- [16] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke, editors. *The Open Problems Project*, <http://maven.smith.edu/~orourke/TOPP/>, May 2006.
- [17] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.E. Ball, T.L. Magnanti, CL. Monma, , and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Sciences: Network Routing*, volume 8, pages 35–139. North-Holland, Amsterdam, 1994.
- [18] K. A. Dowsland and W. B. Dowsland. Packing problems. *European Journal of Operational Research*, 56:2–14, 1992.
- [19] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.
- [20] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution: Typology and Bibliography*. (eds. Mueller, W.A. and Schuster, P.) Springer-Verlag, New York, 1992.
- [21] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and packing. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 393–412. Wiley, 1997.
- [22] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.
- [23] S. P. Fekete and J. Schepers. On more-dimensional packing I: Modeling. *submitted to: Discrete Applied Mathematics*, 1997.
- [24] S. P. Fekete and J. Schepers. On more-dimensional packing II: Bounds. *submitted to: Discrete Applied Mathematics*, 1997.

- [25] S. P. Fekete and J. Schepers. On more-dimensional packing III: Exact Algorithms. *submitted to: Discrete Applied Mathematics*, 1997.
- [26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of \mathcal{NP} -Completeness*. Freeman, San Francisco, 1979.
- [27] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.
- [28] P. Gilmore and R. Gomory. Multistage cutting problems of two and more dimensions. *Operations Research*, 13:94–119, 1965.
- [29] P. Gilmore and R. Gomory. The theory and computation of knapsack functions. *Operations Research*, 15:1045–1075, 1967.
- [30] GLPK. GNU Linear Programming Kit.
<http://www.gnu.org/software/glpk>.
- [31] E. Hopper and B. C. H. Turton. A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems. *Artificial Intelligence Review*, 16:257–300, December 2001.
- [32] E. Hopper and B. C. H. Turton. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. *European Journal of Operations Research*, 128:34–137, 2001.
- [33] E. Horowitz and S. Sahni. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM*, 21(2):277–292, 1974.
- [34] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975.
- [35] G. Jeong, K. Lee, S. Park, and K. Park. A branch-and-price algorithm for the Steiner tree packing problem. *Computers and Operations Research*, 29(3):221–241, 2002.
- [36] L. V. Kantorovic. Mathematical methods of organizing and planning production. *Management Science*, 6:363–422, 1960. (in Russian 1939).
- [37] V. Klee and G. J. Minty. How Good is the Simplex Algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press Inc., New York, 1972.
- [38] A. H. Land and A. G. Doig. An Automatic Method for Solving Discrete Programming Problems. *Econometrica*, 28:497–520, 1960.

- [39] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An Algorithm for the Traveling Salesman Problem. *Operations Research*, 11:972–989, 1963.
- [40] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, September 2002.
- [41] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357, 1999.
- [42] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, 2002.
- [43] A. Lodi, S. Martello, and D. Vigo. Models and Bounds for Two-Dimensional Level Packing Problems. *Journal of Combinatorial Optimization*, 8:363–379, 2004.
- [44] O. Marcotte. An instance of the cutting stock problem for which the rounding property does not hold. *Operations Research Letters*, 4:239–243, 1986.
- [45] S. Martello, M. Monaci, and D. Vigo. An Exact Approach to the Strip-Packing Problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.
- [46] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
- [47] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.
- [48] OR-Library. Operations Research Library.
<http://www.brunel.ac.uk/depts/ma/research/jeb/info.html>.
- [49] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.
- [50] J. Rietz, G. Scheithauer, and J. Terno. Families of non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics*, 121(1-3):229–245, 2002.
- [51] J. Rietz, G. Scheithauer, and J. Terno. Tighter bounds for the gap and non-IRUP constructions in the one-dimensional cutting stock problem. *Optimization*, 6:927–963, 2002.

- [52] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland Publishing Company, 1981.
- [53] M. W. P. Savelsbergh. A Branch-and-Price Algorithm for the Generalized Assignment Problem. *Operations Research*, 45:831–841, 1997.
- [54] G. Scheithauer and J. Terno. A branch and bound algorithm for solving one-dimensional cutting stock problems exactly. *APLICACIONES MATEMATICAE*, 23:151–167, 1995.
- [55] G. Scheithauer and J. Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Operations Research Letters*, 20:93–100, 1997.
- [56] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience, 1986.
- [57] P. H. Vance. Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound. *Computational Optimization and Applications*, 3:111–130, 1994.
- [58] P. H. Vance. Branch-and-Price Algorithms for the One-Dimensional Cutting Stock Problem. *Computational Optimization and Applications*, 9:211–228, March 1998.
- [59] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45:188–200, 1997.
- [60] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86:565–594, Dec 1999.
- [61] F. Vanderbeck. On Dantzig-Wolfe Decomposition in Integer Programming and ways to Perform Branching in a Branch-and-Price Algorithm. *Operations Research*, 48(1):111–128, 2000.
- [62] F. Vanderbeck and L. A. Wolsey. An exact algorithm for IP Column generation. *Operations Research Letters*, 19:151–159, 1996.
- [63] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
- [64] A. L. Vignatti. Aproximação e Compartilhamento de Custos em Projetos de Redes. Master’s thesis, Universidade Estadual de Campinas, 2006.

- [65] G. Wäscher, H. Haußner, and H. Schumann. An Improved Typology of Cutting and Packing Problems. *Working Paper Series*, 2004. Faculty of Economics and Management, Otto von Guericke University, Magdeburg.
- [66] L. A. Wolsey. *Integer programming*. Wiley-Interscience, New York, NY, USA, 1998.
- [67] Xpress-MP®. Dash optimization.
<http://www.dashoptimization.com>.

Índice Remissivo

Algoritmo

- BFDH, 38, 51, 56
- FFDH, 37, 51, 56
- Mochila Gulosa, 40, 50, 56
- Next-Fit, 35
- NFDH, 36, 51, 56
- Programação Dinâmica para Mochila, 18, 40, 48

Algoritmos Aproximados, 16

- Branch-and-Bound, 21, 28
- Branch-and-Price, 22, 31, 45, 51

Conjuntos Independentes, 50

Decomposição de Dantzig-Wolfe, 23, 44

Envoltória Convexa, 23–25

Formulação

- do Problema da Mochila, 9
- do Problema de Corte Unidimensional, 20
- do Problema de Empacotamento Bidimensional em Faixas, 43
- do Problema de Empacotamento Unidimensional, 24
- Mestre, 24, 26, 27, 43

Geração de Colunas, 21, 33, 44, 48, 64

Limitantes, 30, 43, 50, 55

Pricing, 21

Problema

- da Mochila, 9, 18, 39, 45, 48
- de Carregamento de Paletes, 14
- de Corte Bidimensional, 12
- de Corte Unidimensional, 10, 20
- de Empacotamento Bidimensional, 12
- de Empacotamento Bidimensional em Faixas, 12, 15, 42
- de Empacotamento Unidimensional, 10, 24, 54
- do Padrão de Peso Máximo, 44
- Programação Dinâmica, 17
- Problema da Mochila, 18, 48
- Programação Linear, 18
- Proposição de Ryan e Foster, 46, 47
- Ramificação, 30, 33, 45
- Árvore de, 30, 52, 61, 73
- Restrição de Aresta, 49, 61, 65
- Simplex, 19, 21