Algoritmos para Problemas de Corte de Guilhotina Bidimensional

Glauber Ferreira Cintra

Tese apresentada ao
Instituto de Matemática e Estatística da
Universidade de São Paulo para obtenção do
grau de Doutor em Ciência da Computação

Área de Concentração: Ciência da Computação
Orientadora: Profa. Dra. Yoshiko Wakabayashi

Durante a elaboração deste trabalho o autor recebeu apoio financeiro do CNPq

—São Paulo, abril de 2004—

Algoritmos para Problemas de
Corte de Guilhotina Bidimensional

Glauber Ferreira Cintra

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por Glauber Ferreira Cintra e aprovada pela banca examinadora.

Banca examinadora:

- Profa. Dra. Yoshiko Wakabayashi (Orientadora) – IME-USP
- Prof. Dr. Ernesto G. Birgin – IME-USP
- Prof. Dr. Flávio K. Miyazawa – IC-UNICAMP
- Prof. Dr. Horácio H. Yanasse – LAC-INPE
- Prof. Dr. Sóstenes L. S. Lins – DMAT-UFPE

—São Paulo, abril de 2004—

à minha esposa Fabiana,
aos meus filhos Sarah e André
e aos meus pais Lino e Maria

Agradecimentos

Muitas pessoas ajudaram-me durante o doutorado e aproveito este espaço para externar meu sincero agradecimento.

Agradeço aos colegas do IME-USP que me incentivaram e com que compartilhei bons momentos. Em especial, agradeço ao Adriano Rodrigues, Antônio Basile, Estela Rodrigues, Eugênio Nassu, Fábio Martinez, Gordana Manic, Liliane Salgado, Luciano Silva, Marcelo Lauretto, Orlando Lee e Rogério Brito.

Aos administradores da rede do IME-USP, em especial ao Alex Camargo, por mantê-la em funcionamento (*quase*) contínuo, o que permitiu realizar os testes computacionais apresentados nesta tese. Expresso também meus agradecimentos à *Dash Optimization, Inc*, que nos cedeu gratuitamente o software *Xpress-MP* como parte de seu *Programa de Parceiros Acadêmicos*, e que utilizamos na implementação de diversos algoritmos.

Aos professores *Carlos Eduardo Ferreira, Cristina Gomes Fernandes, Hernán Astudillo, José Augusto Ramos Soares* e *José Coelho de Pina Júnior*, que contribuíram para o meu aperfeiçoamento através das disciplinas que cursei durante o doutorado, e aos professores *Carlos Eduardo Ferreira* e *Flávio Keidi Miyazawa* que participaram como membros da banca na minha qualificação e cujas sugestões foram valiosas para a consumação deste trabalho.

À professora *Yoshiko Wakabayashi*, por suas demonstrações de confiança e carinho e pela forma dedicada e paciente com que me orientou.

Aos meus pais, *Lino* e *Maria*, que plantaram as sementes que agora estou colhendo.

Agradeço em especial à *Fabiana*, minha esposa, e aos meus filhos *Sarah* e *André*, pelo seu apoio, compreensão incentivo e amor.

Algoritmos para Problemas de Corte de Guilhotina Bidimensional

Glauber Ferreira Cintra

Resumo da tese apresentada ao IME-USP como parte dos requisitos necessários para a obtenção do título de Doutor em Ciências

Resumo

Muitas indústrias têm como desafio encontrar soluções mais econômicas possíveis para o problema de cortar objetos grandes visando a produção de objetos menores de dimensões especificadas, ou o problema de empacotar uma coleção de objetos pequenos dentro de objetos grandes. Tais problemas são chamados de problemas de corte e empacotamento, e são, em geral, $\mathcal{NP}$-difíceis. Em muitas aplicações, os objetos grandes (placas) e os objetos pequenos (itens) têm apenas duas dimensões relevantes e possuem a forma retangular. Além disso, é comum a restrição de que os cortes em cada objeto sejam de guilhotina, isto é, estes devem ser paralelos a um de seus lados e se estender desde um lado do objeto até o lado oposto; problemas desse tipo são chamados de problemas de corte de guilhotina bidimensional. Algoritmos para tais tipos de problemas constituem o tema central desta tese.

Investigamos o problema de corte de estoque bidimensional com demandas (PCED_2) (um caso mais geral em que os cortes não precisam ser de guilhotina) e introduzimos o conceito de padrões semi-homogêneos. Fazendo uso de tais padrões desenvolvemos um algoritmo polinomial cuja razão de aproximação absoluta é 4, e mostramos que esta razão é justa. Ainda utilizando padrões semi-homogêneos, desenvolvemos um algoritmo que resolve uma variante do PCED_2 na qual as placas e os itens são quadrados. Provamos que este algoritmo tem razão de aproximação assintótica entre 2,4166 e 2,6875. Até onde sabemos, estes são os primeiros algoritmos de aproximação propostos para tais problemas. Desenvolvemos ainda um algoritmo para o problema de corte de estoque bidimensional binário com rotações e provamos que esse algoritmo possui razão de aproximação assintótica não maior que 4.

Utilizando a fórmula de recorrência proposta por Beasley e os pontos de discretização definidos por Herz, desenvolvemos um algoritmo pseudo-polinomial para o problema de corte de guilhotina bidimensional com valor (PCGV_2) baseado em programação dinâmica. Chamamos tal algoritmo de PCGV_2PD. Este algoritmo também resolve uma variante do PCGV_2 na qual os itens podem sofrer rotações ortogonais. Apresentamos também um algoritmo baseado em enumeração explícita e em programação dinâmica para calcular os pontos de discretização. Mostramos que, se os itens não são muito pequenos em relação ao tamanho das placas, então o algoritmo PCGV_2PD requer tempo polinomial. Implementamos o PCGV_2PD e resolvemos todas as instâncias do PCGV_2 encontradas na OR–LIBRARY. Destacamos que para uma destas instâncias (mencionada há duas décadas) não se conhecia uma solução ótima.

Aplicamos o método de geração de colunas para o problema de corte de guilhotina bidimensional com demandas (PCGD_2), utilizando o $\mathrm{PCGV}_2\mathrm{PD}$ para gerar as colunas. Mostramos também como aplicar esta técnica para resolver o PCGD_2 com rotações (PCGD_2^r). Introduzimos a idéia de perturbar as instâncias residuais como forma de obter soluções de melhor qualidade. Obtivemos assim quatro algoritmos heurísticos. Resolvemos diversas instâncias do PCGD_2 e do PCGD_2^r com estes quatro algoritmos, tendo obtido soluções ótimas ou quase-ótimas em todos os casos.

Estudamos ainda uma variante do PCGD_2 na qual as placas podem não ser idênticas ($\mathrm{PCGD}_2\mathrm{V}$) e a sua versão com rotações ($\mathrm{PCGD}_2^r\mathrm{V}$). Tais problemas foram muito pouco abordados na literatura. Adaptamos os métodos descritos para o PCGD_2 e obtivemos quatro algoritmos para estes problemas. Resolvemos diversas instâncias do $\mathrm{PCGD}_2\mathrm{V}$ e do $\mathrm{PCGD}_2^r\mathrm{V}$ com estes algoritmos, tendo obtido soluções quase-ótimas em todos os casos.

Verificamos que os algoritmos que propomos para o PCGV_2, PCGD_2, $\mathrm{PCGD}_2\mathrm{V}$ e suas versões com rotações apresentaram um bom desempenho, em termos de tempo e de qualidade das soluções encontradas. Esses experimentos foram feitos com diversas instâncias de pequeno e médio porte. Tais evidências empíricas mostram que estes algoritmos parecem ser apropriados para resolver instâncias associadas a situações reais.

Palavras-chave: problemas de corte e empacotamento, problemas de corte de estoque, empacotamento bidimensional, corte de guilhotina, algoritmos de aproximação, razão assintótica, programação dinâmica, geração de colunas

Orientadora da Tese: Yoshiko Wakabayashi

Algorithms for Two-dimensional Guillotine Cutting Problems

Glauber Ferreira Cintra

Abstract of the thesis presented to IME-USP in partial fulfillment of the requirements for the degree of Doctor of Science

Abstract

Many industries face the challenge of finding solutions that are the most economical for the problem of cutting large objects to produce specified smaller objects, or the problem of packing a collection of small objects into larger ones. These problems are called cutting and packing problems, and are generally $\mathcal{NP}$-hard. Very often, the large objects (bins) and the small objects (items) have only two relevant dimensions and have rectangular shape. Besides that, a usual restriction for cutting problems is that in each object we may use only guillotine cuts, that is, cuts that are parallel to one of the sides of the object and go from one side to the opposite one; problems of this type are called two-dimensional guillotine cutting problems. This thesis focuses on algorithms for such problems.

We investigate the two-dimensional cutting problem with demands (PCED_2) (a more general version in which the cuts must be orthogonal but are not required to be guillotine cuts) and we introduce the concept of semi-homogeneous patterns. Making use of such patterns we exhibit a polynomial 4-approximation algorithm for this problem, and we prove that this absolute performance ratio is tight. We also use such patterns to design an algorithm for a variant of PCED_2, in which the bins and the items are squares. We prove that this algorithm has an asymptotic performance ratio between 2,4166 and 2,6875. To our knowledge these are the first approximation algorithms proposed for these problems. We also exhibit an algorithm for the two-dimensional binary cutting problem with rotations and prove that its asymptotic performance ratio is at most 4.

Using the recurrence relation proposed by Beasley and the idea of discretization points defined by Herz, we design a pseudo-polynomial time algorithm for the two-dimensional guillotine cutting problem with value (PCGV_2), based on dynamic programming. We call this algorithm PCGV_2PD. This algorithm also solves a variant of PCGV_2 in which all items can be rotated orthogonally. We also present an algorithm based on explicit enumeration and dynamic programming to calculate the discretization points. We show that if the items are not so small compared to the size of the bins, then algorithm PCGV_2PD requires polynomial time. We have implemented PCGV_2PD and we have solved all instances of PCGV_2 found at OR–LIBRARY. We remark that no optimal solution to one of these instances was known (this instance appeared two decades ago).

We present a column generation based algorithm for the two-dimensional guillotine cutting

problem with demands ($PCGD_2$) that makes use of $PCGV_2PD$ to generate the columns. We also show how to apply this approach to solve $PCGD_2$ with rotations ($PCGD_2^r$). We introduce the idea of perturbing the residual instances as an strategy to look for better solutions. We have obtained this way four heuristic algorithms. We have solved many instances of $PCGD_2$ and $PCGD_2^r$ with these algorithms, and obtained optimal or quasi-optimal solutions in all cases.

We also study a variant of $PCGD_2$ in which all bins need not be identical ($PCGD_2V$) and its version where orthogonal rotations are allowed ($PCGD_2^rV$). In the literature we did not find many references to these problems. We adapt the methods we described for $PCGD_2$ and obtained four algorithms for these problems. We have solved many instances of $PCGD_2V$ and of $PCGD_2^rV$ with these algorithms and obtained quasi-optimal solutions in all cases.

We have observed that the algorithms proposed for $PCGV_2$, $PCGD_2$, $PCGD_2V$ and their variants in which rotations are allowed performed well, in terms of time and quality of the obtained solutions. These experiments were run on many small and medium size instances. This empirical evidence indicates that these algorithms seem to be suitable for solving real-world instances.

Keywords: cutting and packing problems, cutting stock problem, two-dimensional packing, guillotine cut, approximation algorithms, asymptotic ratio, dynamic programming, column generation

Thesis advisor: Yoshiko Wakabayashi

Índice

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Existe uma grande diversidade de situações em que nos deparamos com o seguinte desafio: precisamos cortar objetos grandes, produzindo objetos menores, de forma a obter ganhos em termos econômicos. Em tais situações estão envolvidos problemas de corte. Em outras situações, temos um desafio bastante similar que consiste em colocar uma coleção de objetos pequenos dentro de objetos grandes, que chamaremos de *recipientes*, obtendo algum tipo de vantagem econômica. Nestes caso, temos associado um problema de empacotamento.

Embora na prática os problemas de corte sejam distintos dos problemas de empacotamento, para fins de formulação e resolução podemos considerar estes dois tipos de problemas como sendo análogos. Queremos dizer com isto que para todo problema de corte existe um problema de empacotamento análogo, e vice-versa. Um problema de corte e o seu problema de empacotamento análogo podem ser formulados e resolvidos da mesma maneira. Neste capítulo vamos nos referir genericamente a estes problemas como problemas de corte e empacotamento. Nos capítulos seguintes vamos designar cada um dos problemas tratados como sendo ou de corte ou de empacotamento.

Em muitas situações, os objetos grandes e os objetos pequenos têm apenas duas dimensões relevantes e possuem a forma retangular. Além disso, é comum a restrição de que os cortes feitos em cada objeto grande têm que ser paralelos a um de seus lados e se estender desde um lado do objeto até o lado oposto. Chamamos este tipo de corte de *corte de guilhotina*.

Os problemas de corte e empacotamento que envolvem as restrições que citamos no parágrafo anterior são chamados genericamente de problemas de corte de guilhotina bidimensional. Tais problemas constituem o assunto principal desta tese. Abordamos também alguns outros problemas de corte e empacotamento bidimensional.

A pesquisa que serviu de base para a elaboração deste texto teve dois objetivos principais: o primeiro foi conhecer as principais técnicas de resolução propostas na literatura para alguns casos particulares dos problemas de corte de guilhotina bidimensional. Explicamos tais casos particulares e apresentamos nesta tese diversos algoritmos de aproximação e algoritmos exatos propostos para eles.

O outro objetivo foi propor métodos de resolução que apresentassem um bom desempenho, em termos de tempo e de qualidade das soluções encontradas ao resolver instâncias consideradas representativas daquelas que surgem com maior frequência em situações reais. Desenvolvemos algoritmos exatos baseados em programação dinâmica e em métodos puramente combinatórios. Propusemos também diversos métodos heurísticos baseados em programação linear e nos algoritmos exatos a que nos referimos. Implementamos e testamos estes algoritmos e métodos heurísticos, tendo obtido resultados animadores e de certa forma surpreendentes.

Desenvolvemos também alguns algoritmos de aproximação, fornecendo suas respectivas razões de aproximação. Até onde sabemos, alguns deles são os primeiros algoritmos de aproximação desenvolvidos para certas variantes do problema de corte de estoque bidimensional.

Na próxima seção procuramos despertar o interesse do leitor pelo estudo dos problemas de corte e empacotamento. Apresentamos motivações de caráter teórico, citando algumas conexões com outras áreas de pesquisa e discutindo a complexidade computacional intrínseca a estes problemas. Além disso, apresentamos alguns resultados de inaproximabilidade relacionados com alguns problemas de corte e empacotamento. Fornecemos também motivações de ordem prática, apresentando diversas situações em que problemas de corte e empacotamento surgem naturalmente.

Supomos na seção seguinte que o leitor possui conhecimentos básicos de complexidade computacional e de algoritmos de aproximação e esteja familiarizado com problemas de corte e empacotamento. Se este não for o caso, talvez seja vantajoso ler primeiramente os capítulos 2 e 3 antes de prosseguir com a leitura desta introdução.

Na última seção deste capítulo, discorremos sobre a organização desta tese, resumindo os principais assuntos que serão tratados nos capítulos seguintes.

1.1 Motivação

Devido à grande variedade de situações do mundo real onde surgem problemas de corte e empacotamento, um número crescente de pesquisadores de diversas áreas, tais como computação, economia, engenharia e matemática, têm se dedicado ao estudo destes problemas. Nestes estudos têm sido aplicadas diversas técnicas de resolução de problemas tais como *programação linear*, *programação dinâmica*, *branch-and-bound*, etc.

Além disso, avanços significativos têm sido obtidos em outras áreas do conhecimento como fruto de pesquisas envolvendo problemas de corte e empacotamento. Destacamos em especial os avanços obtidos nas áreas de teoria da complexidade computacional e algoritmos de aproximação.

O próprio termo *approximation algorithm* foi introduzido por Johnson [Joh74] ao propor algoritmos para o problema de empacotamento unidimensional. Os primeiros resultados provando a inexistência, sob a hipótese de que $\mathcal{P} \neq \mathcal{NP}$, de algoritmos *on-line* com razão de aproximação menor que certas constantes, envolveram problemas de empacotamento. Além disso, os primeiros PAAS e FPAAS para problemas fortemente $\mathcal{NP}$-completos foram desenvolvidos para o problema de empacotamento unidimensional [FL81, KK82].

Os problemas de corte e empacotamento costumam ser fáceis de entender e formular. No entanto, sua aparente simplicidade costuma esconder sua natureza complexa, em termos computacionais. Queremos dizer com isto que a maioria dos problemas de corte e empacotamento abordados na literatura não podem ser resolvidos por algoritmos polinomiais, a menos que $\mathcal{P} = \mathcal{NP}$.

Discutimos a seguir a complexidade computacional dos problemas que abordaremos nesta tese e apresentamos alguns resultados de inaproximabilidade relativos a problemas de corte e empacotamento.

1.1.1 Complexidade Computacional de Problemas de Corte e Empacotamento

Muitos problemas de corte e empacotamento têm como casos particulares dois problemas combinatórios bem conhecidos: o problema da mochila inteira (PMI) e o problema da 3-partição (P3P).

O PMI consiste em: dada uma mochila de capacidade C e uma lista de m tipos de

objetos, onde cada objeto do tipo i possui peso p_i e valor v_i, determinar os inteiros e não-negativos x_i $(i = 1, \ldots, m)$, tais que $\sum_{i=1}^{m} p_i x_i \leq C$ e que maximizam $\sum_{i=1}^{m} v_i x_i$. Cada variável x_i indica quantos objetos do tipo i devem ser colocados na mochila. O objetivo então é colocar objetos dentro da mochila, sem exceder sua capacidade, fazendo com que a soma dos valores destes objetos seja a maior possível. Lueker [Lue75] mostrou que este problema é $\mathcal{NP}$-difícil.

O P3P é o seguinte problema: dado um número inteiro B e um conjunto A com $3m$ elementos, onde cada elemento $a \in A$ possui valor $\frac{B}{4} < v_a < \frac{B}{2}$, e $\sum_{a \in A} v_a = mB$, queremos particionar A em subconjuntos disjuntos $A_1, \ldots, A_m$ tais que $\sum_{a \in A_i} v_a = B$ $(i = 1, \ldots, m)$. Garey e Johnson [GJ75] provaram que o P3P é $\mathcal{NP}$-difícil.

Na Tabela 1.1 relacionamos os problemas de corte e empacotamento[1] para os quais propusemos algoritmos, com exceção do problema de empacotamento de quadrados em quadrados com demandas (PEQD), que também tratamos. Para cada problema citado na tabela, indicamos um de seus casos particulares que é $\mathcal{NP}$-difícil. Conforme indicamos, o PCEB_2 e o PCGD_2 têm como caso particular o problema de empacotamento de *bins* binário (PEBB), que por sua vez é $\mathcal{NP}$-difícil por conter o P3P como caso especial.

Problema de Corte e Empacotamento	Caso Particular
Problema de corte de estoque bidimensional binário (PCEB_2)	PEBB
Problema de corte de estoque bidimensional com demandas (PCED_2)	PCEB_2
Problema de corte de guilhotina bidimensional com demandas (PCGD_2)	PEBB
PCGD_2 com rotações (PCGD_2^r)	PCGD_2
PCGD_2 com placas de dimensões variadas (PCGD_2V)	PCGD_2
PCGD_2V com rotações (PCGD_2^rV)	PCGD_2V
Problema de corte de guilhotina bidimensional com valor (PCGV_2)	PMI
PCGV_2 com rotações (PCGV_2^r)	PCGV_2

Tabela 1.1: Problemas de corte e empacotamento $\mathcal{NP}$-difíceis.

Desenvolvemos ainda um algoritmo de aproximação para o PEQD, que também é $\mathcal{NP}$-difícil [LTW+90, FMW99]. Sendo assim, todos os problemas que vamos abordar nesta tese são $\mathcal{NP}$-difíceis.

[1] A definição formal destes problemas é apresentada nos capítulos 3 e 4.

1.1.2 Inaproximabilidade de Problemas de Corte e Empacotamento

Sabe-se que algumas variantes dos problemas de corte e empacotamento não são aproximáveis, em termos absolutos, abaixo de determinadas constantes, supondo que $\mathcal{P} \neq \mathcal{NP}$. Por exemplo, Garey Johnson [GJ79] provaram que não existe algoritmo de aproximação para o PEBB com razão de aproximação absoluta menor que 1,5. Brown, Baker e Katseff [BBK82] mostraram que qualquer algoritmo *on-line* para o problema de empacotamento em faixa binário (PEFB) tem que ter razão de aproximação absoluta maior ou igual a 2.

Existem ainda resultados envolvendo inaproximabilidade de problemas de corte e empacotamento, em termos assintóticos. Por exemplo, Van Vliet [Vli92] provou que não existe algoritmo *on-line* para o PEBB com razão de aproximação assintótica menor que 1,5401. Csirik e Woeginger [CW97] demonstraram que qualquer algoritmo *on-line* para o PEFB, baseado em níveis, tem que ter razão de aproximação assintótica maior ou igual a 1,691. Sabe-se ainda que qualquer algoritmo *on-line* para o PCEB_2 tem que ter razão de aproximação assintótica maior ou igual a 1,907 [BvW96].

1.1.3 Aplicações Práticas para Problemas de Corte e Empacotamento

O interesse por estes problemas é em parte explicado por sua grande aplicabilidade prática, especialmente nas indústrias. Pequenas melhorias nos processos que envolvem corte e empacotamento podem levar a ganhos substanciais, dependendo da escala de produção, e representar uma vantagem decisiva na competição com outras empresas do setor.

Podemos citar os processos industriais envolvendo corte de bobinas de tecido, de barras de aço, alumínio e de canos. Tais processos estão presentes na indústria têxtil, de construção civil e siderúrgica. Em todos estes processos estão envolvidos problemas de corte unidimensional.

Temos ainda processos envolvendo corte de chapas de metal e madeira, lâminas de vidro e fibra de vidro, peças de couro e carpete. Estes processos ocorrem na indústria metalúrgica, moveleira, vidraceira e na indústria da moda. Estes processos estão associados a problemas de corte bidimensional.

As indústrias alimentícias, farmacêuticas, de cosmésticos, etc., precisam empacotar seus produtos, geralmente em caixas de papelão, e guardá-los em armazéns. As empresas

de transporte rodoviário, ferroviário, marítimo e aéreo precisam colocar as cargas em contêineres e caminhões-baú. Estes contêineres muitas vezes precisam ser empilhados em navios. Em todas estas situações surgem problemas de empacotamento bi e tridimensional.

Existe ainda uma infinidade de aplicações práticas para os problemas de corte e empacotamento. Por exemplo, determinar como arranjar os processos na memória principal de um computador de modo a diminuir a necessidade de fazer *swap* entre a memória principal e a memória secundária envolve um problema de empacotamento. Um outro exemplo seria o planejamento da sequência de exibição de anúncios durante os intervalos comerciais na programação de uma emissora de rádio ou televisão. Enfim, existe um largo espectro de aplicações práticas para os problemas de corte e empacotamento.

1.2 Organização da Tese

Este texto está organizado da seguinte maneira. Inicialmente, no **Capítulo 2**, introduzimos alguns conceitos sobre complexidade computacional e algoritmos de aproximação. Ademais, estabelecemos parte da notação que será utilizada nos capítulos seguintes.

Descrevemos no **Capítulo 3** as principais características dos problemas de corte e empacotamento e mencionamos a classificação proposta por Dyckhoff [Dyc90] para esses problemas. Além disso, definimos formalmente os principais problemas que iremos abordar no restante da tese.

A seguir, abordamos no **Capítulo 4** diversos algoritmos de aproximação encontrados na literatura, propostos para problemas de empacotamento em uma ou duas dimensões. Ênfase especial é dedicada aos algoritmos que servem de base ou que são utilizados como sub-rotina nos algoritmos que vamos propor nos capítulos subsequentes.

No **Capítulo 5** abordamos o problema de corte de estoque bidimensional com demandas (PCED_2). Desenvolvemos um algoritmo de aproximação para o PCED_2 que utiliza apenas padrões homogêneos, mas cuja razão de aproximação é infinita. Introduzimos o conceito de blocos homogêneos e padrões semi-homogêneos e mostramos como utilizá-los no desenvolvimento de um algoritmo de aproximação para o PCED_2. Provamos que este algoritmo tem razão de aproximação absoluta igual a 4 e que esta razão é justa.

Apresentamos ainda um algoritmo de aproximação para a variante do PCED_2 em que as placas e os itens são quadrados. Mostramos que tal algoritmo tem razão de aproximação assintótica não maior que 2,6875. Propomos também um algoritmo de aproximação bastante simples para o problema de corte de guilhotina bidimensional binário com ro-

tações e mostramos que sua razão de aproximação assintótica não é maior que 4. Este algoritmo é usado como sub-rotina em dois outros algoritmos que desenvolvemos para o $PCGD_2^r$.

Em seguida, no **Capítulo 6**, introduzimos as fórmulas de recorrência propostas por Gilmore e Gomory [GG65], e por Beasley [Bea85a], para o problema de corte de guilhotina bidimensional com valor ($PCGV_2$). Mostramos que tais fórmulas podem ser calculadas usando-se algoritmos pseudo-polinomiais baseados em programação dinâmica. Definimos os pontos de discretização de Herz [Her72] e mostramos como calculá-los utilizando programação dinâmica e *branch-and-bound*.

Utilizando a fórmula de recorrência proposta por Beasley e os pontos de discretização de Herz, apresentamos um algoritmo baseado em programação dinâmica para resolver o $PCGV_2$. Indicamos ainda como utilizar este algoritmo para resolver a variante do $PCGV_2$ na qual os itens podem sofrer rotações ortogonais. Os resultados computacionais obtidos ao resolver diversas instâncias de teste da literatura são apresentados.

No **Capítulo 7** abordamos o problema de corte de guilhotina bidimensional com demandas ($PCGD_2$). Introduzimos o método de geração de colunas e mostramos como implementá-lo de modo a obter soluções quase-ótimas para o $PCGD_2$. Introduzimos ainda a idéia de perturbar as instâncias residuais de forma a obter soluções de melhor qualidade. Obtemos assim dois algoritmos para o $PCGD_2$, ambos baseados na técnica de geração de colunas. Tratamos ainda da variante do $PCGD_2$ na qual rotações são permitidas ($PCGD_2^r$). Os resultados computacionais obtidos ao resolver diversas instâncias de teste são apresentados e discutidos.

Estudamos no **Capítulo 8** a variante do $PCGD_2$ onde as placas podem não ser idênticas. Chamamos tal variante de problema de corte de guilhotina bidimensional com demandas e com placas de dimensões variadas ($PCGD_2V$). Adaptamos as técnicas introduzidas no Capítulo 7, obtendo dois algoritmos para o $PCGD_2V$.

Investigamos também a variante do $PCGD_2V$ na qual rotações são permitidas, que chamamos de $PCGD_2^rV$. Adaptamos para esta variante os algoritmos propostos para o $PCGD_2V$. Apresentamos e discutimos os resultados computacionais obtidos ao resolver diversas instâncias de teste.

Finalmente, no **Capítulo 9** tecemos algumas considerações sobre nossas contribuições e sobre a implementação dos algoritmos por nós propostos. Discutimos também alguns possíveis desdobramentos de nossa pesquisa.

Incluímos também um **Apêndice** onde apresentamos os detalhes das instâncias de

teste utilizadas nos capítulos 6, 7 e 8, e fornecemos as soluções encontradas para estas instâncias pelos algoritmos que desenvolvemos, comparando-as com os limites inferiores obtidos para os valores de suas soluções ótimas. Por fim, no última página listamos os principais problemas estudados neste tese e suas respectivas siglas.

Capítulo 2

Preliminares

2.1 Introdução

Neste capítulo introduzimos alguns conceitos sobre complexidade computacional e algoritmos de aproximação. Ademais, estabelecemos parte da notação e da terminologia que será utilizada nos capítulos seguintes. Ao abordar programação linear, nos capítulos 7 e 8, vamos adotar os conceitos e definições do livro *Linear Programming* [Chv80].

2.2 Complexidade Computacional

Basicamente, utilizaremos os conceitos sobre complexidade computacional explicados em [GJ79] e [CLRS01]. Convém no entanto chamar atenção para alguns termos que utilizaremos com frequência nesta tese.

Em geral, estaremos interessados apenas na complexidade de tempo dos algoritmos apresentados nesta tese. Assim, quando usarmos o termo complexidade deve ficar subentendido que estamos nos referindo a complexidade de tempo. Vamos descrever a complexidade dos algoritmos através de funções no tamanho das instâncias, utilizando a notação $\mathcal{O}$.

Dizemos que um algoritmo para um problema P é *polinomial*, se a quantidade de passos requeridos pelo algoritmo para resolver qualquer instância de P está limitada por um polinômio no tamanho da entrada. Ao comentar os algoritmos descritos nesta tese, frequentemente indicaremos sua complexidade de tempo, sem no entanto fornecer uma demonstração rigorosa.

2.3 Algoritmos de Aproximação

Os conceitos aqui explicados estão baseados em [GJ79] e [FMC+01]. Dado um problema de minimização $\mathcal{P}$, denotamos por $\mathrm{OPT}(I)$ o valor de uma solução ótima de uma instância I de $\mathcal{P}$. Dado um algoritmo $\mathcal{A}$ para $\mathcal{P}$, denotamos por $\mathcal{A}(I)$ o valor obtido pelo algoritmo $\mathcal{A}$ para a instância I de $\mathcal{P}$.

Dizemos que um algoritmo $\mathcal{A}$ é um *algoritmo de aproximação* para um problema de minimização $\mathcal{P}$ se $\mathcal{A}$ é polinomial e existem funções α e β tais que

$$\mathcal{A}(I) \leq \alpha\ \mathrm{OPT}(I) + \beta, \forall\ I \in \mathcal{P}. \tag{2.1}$$

A *razão de aproximação absoluta* do algoritmo $\mathcal{A}$, denotada por $R_{\mathcal{A}}$, é definida da seguinte forma:

$$R_{\mathcal{A}} = inf\{r \mid \frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} \leq r, \forall\ I \in \mathcal{P}\}.$$

Se em (2.1) temos que $\beta = 0$, então dizemos que o algoritmo $\mathcal{A}$ é uma α-aproximação absoluta para o problema $\mathcal{P}$ e que α é *uma* razão de aproximação absoluta do algoritmo $\mathcal{A}$. Dizemos que a razão α é *justa* se para todo $\epsilon > 0$, existe uma instância $I \in \mathcal{P}$ tal que $\frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} > \alpha - \epsilon$. Neste caso, temos que $R_{\mathcal{A}} = \alpha$.

Definimos a *razão de aproximação assintótica* do algoritmo $\mathcal{A}$, denotada por $R^{\infty}_{\mathcal{A}}$, da seguinte maneira[1]:

$$R^{\infty}_{\mathcal{A}} = \limsup_{n\to\infty} \big(max\{\frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} \mid I \in \mathcal{P} \text{ e } \mathrm{OPT}(I) = n\}\big).$$

Se em (2.1) temos que β é uma constante, dizemos que o algoritmo $\mathcal{A}$ é uma α-aproximação assintótica para o problema $\mathcal{P}$ e que α é *uma* razão de aproximação assintótica do algoritmo $\mathcal{A}$. Dizemos que a razão α é *justa* se para todo $\epsilon > 0$ e todo $n > 0$, existe uma instância $I \in \mathcal{P}$ tal que $\mathrm{OPT}(I) > n$ e $\frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} > \alpha - \epsilon$. Neste caso, temos que $R^{\infty}_{\mathcal{A}} = \alpha$.

Chamamos de *esquema de aproximação assintótica de tempo polinomial* (*polynomial time asymptotic approximation scheme*), ou simplesmente PAAS, um conjunto de algoritmos $\mathcal{A}$ tal que, para todo $\epsilon > 0$, existe uma constante β e um algoritmo $\mathcal{A}_{\epsilon} \in \mathcal{A}$, polinomial em ϵ e no tamanho da entrada, tal que

[1] Esta definição aparece em [CGJ82].

$$\mathcal{A}_\epsilon(I) \leq (1+\epsilon)\ \mathrm{OPT}(I) + \beta.$$

Chamamos de *esquema de aproximação assintótica de tempo completamente polinomial* (*fully polynomial time asymptotic approximation scheme*), ou simplesmente FPAAS, um conjunto de algoritmos $\mathcal{A}$ tal que, para todo $\epsilon > 0$, existe uma constante β e um algoritmo $\mathcal{A}_\epsilon \in \mathcal{A}$, polinomial em $\frac{1}{\epsilon}$ e no tamanho da entrada, tal que

$$\mathcal{A}_\epsilon(I) \leq (1+\epsilon)\ \mathrm{OPT}(I) + \beta.$$

Todos os algoritmos de aproximação discutidos nesta tese são para problemas de minimização. Por este motivo, não fornecemos as definições correspondentes para problemas de maximização. Tais definições podem ser obtidas em [GJ79].

Capítulo 3

Problemas de Corte e Empacotamento

3.1 Introdução

As peculiaridades dos processos que envolvem corte e empacotamento deram origem a um grande número de variantes do problema, conhecidas na literatura por diversos nomes, tais como problema de corte de estoque unidimensional, problema de empacotamento de *bins*, problema de empacotamento em faixa, problema de carregamento de contêineres, problema da mochila, etc.

Neste capítulo, mencionamos a classificação proposta por Dyckhoff [Dyc90] para os problemas de corte e empacotamento, e em seguida descrevemos com detalhes as variantes que serão tratadas nos capítulos seguintes. Estudar os fundamentos nos quais a classificação de Dyckhoff está baseada pode ajudar o leitor a ter uma visão mais abrangente de muitas das particularidades desses problemas. Não pretendemos, no entanto, discutir exaustivamente estas particularidades, mas sim apresentar as principais características dos problemas de corte e empacotamento, citando algumas variantes como exemplo.

3.2 Tipologia dos Problemas de Corte e Empacotamento

Com o objetivo de sistematizar o estudo dos problemas de corte e empacotamento, Dyckhoff [Dyc90] propôs uma classificação fundamentada na estrutura lógica de tais pro-

blemas e em algumas de suas principais características. Segundo Dyckhoff, os problemas de corte e empacotamento possuem a seguinte estrutura lógica:

- Existem dois grupos básicos de dados, cujos elementos definem objetos geométricos de uma ou mais dimensões. O primeiro grupo de dados contém as dimensões, as quantidades disponíveis e eventualmente outras informações dos objetos grandes, que chamaremos simplesmente de *objetos*, e o segundo grupo define as dimensões, as quantidades requisitadas e eventualmente outras informações dos objetos pequenos, que chamaremos de *itens*.
- O processo de corte (ou empacotamento) produz combinações geométricas dos itens dentro dos objetos. Chamaremos cada possível forma de cortar um objeto de *padrão de corte* (ou simplesmente *padrão*).

A seguir apresentamos as principais características dos problemas de corte e empacotamento. Supomos que o leitor possui uma certa familiaridade com este tipo de problemas, razão pela qual deixamos de definir formalmente as variantes do problema citadas como exemplo nesta seção.

3.2.1 Características dos Problemas de Corte e Empacotamento

A estrutura lógica descrita anteriormente fornece um esquema para sistematizar os problemas de corte e empacotamento. A partir desta sistematização podemos identificar características comuns em problemas que, à primeira vista, parecem não estar relacionados.

Por outro lado, diferenças entre problemas aparentemente similares podem ser detectadas. A classificação proposta por Dyckhoff leva em consideração características geométricas e combinatórias dos objetos, itens e processos de corte e empacotamento. A seguir descrevemos brevemente estas características.

3.2.1.1 Dimensionalidade

Usualmente os objetos e itens são considerados como tendo uma, duas, três ou mais dimensões relevantes. Em muitas aplicações práticas, as dimensões relevantes dos objetos e itens representam comprimento (ou profundidade), largura e altura. Em outras aplicações, as dimensões podem ser peso [Dan57, EC71], tempo [WM82, CGJ78], valores monetários [LS55, MT80, Ste83], espaço de memória RAM [GJ81], etc.

Chamamos de dimensionalidade o número de dimensões necessárias para descrever a geometria dos padrões. Os tipos elementares são: unidimensional, bidimensional, tridimensional e multidimensional (mais de 3 dimensões). Estes quatro tipos elementares de dimensão podem parecer suficientes para classificar os problemas, no entanto, para certos problemas estes tipos elementares não são adequados. Por exemplo, o problema de carregamento de páletes onde a altura do pálete é restrita [Dow85] é considerado como sendo de dimensão "2 + 1", em vez de 3.

3.2.1.2 Medidas de Quantidade

A quantidade de objetos utilizados numa solução pode ser discreta (inteira) ou contínua (fracionária). No primeiro caso contamos o número de objetos utilizados. No caso contínuo, uma das dimensões do objeto é ilimitada, portanto medimos a quantidade utilizada nesta dimensão.

Um exemplo onde a medição é contínua é o problema de empacotamento em faixa. Neste problema temos um objeto de largura fixa e altura ilimitada. Tal problema é considerado como sendo 1,5-dimensional [DKAG85]. Outro exemplo é o problema de empacotamento em altura, onde o objeto tem largura e profundidade fixas e altura ilimitada. Este problema é considerado como sendo 2,5-dimensional.

3.2.1.3 Figura dos Objetos e Itens

Outra característica importante, diretamente relacionada com a dimensionalidade, é a *figura* dos objetos e itens. A figura de um objeto ou item é definida por sua forma geométrica, seu tamanho e sua orientação.

A forma geométrica de um objeto ou item pode ser regular ou irregular. Formas regulares podem ser descritas através de um pequeno conjunto de parâmetros. Na grande maioria dos problemas considerados na literatura os objetos e itens possuem formas regulares, tais como segmentos de reta, retângulos e paralelepípedos. No entanto, figuras de formas irregulares, incluindo formas não-convexas e assimétricas, são comuns em algumas aplicações industriais.

Um fator que frequentemente tem influência na dificuldade de resolver um problema específico é o tamanho dos itens em relação ao tamanho dos objetos [HT90]. Em geral, quanto menor o tamanho dos itens em relação ao tamanho dos objetos, mais difícil é achar uma solução ótima para o problema. Isto ocorre porque, neste caso, a quantidade

de padrões aumenta consideravelmente. Por outro lado, os algoritmos de aproximação para os problemas em que os itens são pequenos quando comparados com os objetos, costumam ter uma melhor razão de aproximação.

No que diz respeito à orientação dos itens em relação aos objetos, podemos distinguir três casos: (a) qualquer orientação é permitida, (b) apenas rotações de 90 graus (em uma ou mais direções) são permitidas e (c) a orientação é fixa (em todas as direções). Neste último caso dizemos que o problema é *orientado*.

3.2.1.4 Sortimento

Indica a variedade de formas geométricas e figuras distintas encontradas no problema. Na indústria de artefatos de metal, por exemplo, é comum os objetos possuírem forma retangular enquanto que os itens possuem formas variadas, incluindo retângulos, círculos, etc. Mesmo quando os objetos e os itens possuem a mesma forma geométrica, diferenças de tamanho e orientação resultam em figuras distintas.

3.2.1.5 Disponibilidade

Esta característica refere-se à quantidade disponível de objetos e itens e à ordem em que podem ser utilizados. A quantidade de objetos e itens pode ser infinita ou finita. Quando lidamos com quantidades finitas, podemos ter muitos ou apenas uns poucos objetos ou itens.

Tanto o problema de empacotamento quanto o problema de corte, na sua forma clássica, pressupõem uma quantidade ilimitada de objetos. Usualmente, no entanto, no problema de empacotamento têm-se poucos itens de cada figura e um grande número de figuras, ao contrário do que ocorre no problema de corte onde têm-se poucas figuras, mas um grande número de itens de cada figura.

Em alguns problemas têm-se apenas um objeto disponível. Desejamos então determinar um padrão que utilize da melhor forma possível este único objeto. O problema de carregamento de páletes do produtor e o problema da mochila são exemplos de problemas caracterizados por utilizar um único objeto.

A ordem em que os objetos e os itens podem ser utilizados é outro aspecto da disponibilidade. Por exemplo, ao empacotar itens que serão utilizados em uma linha de montagem, é preciso levar em consideração a relação de ordem parcial existente entre os itens. Outro exemplo é o carregamento de caminhões-baús, onde os itens têm que ser descarregados

em diferentes pontos de entrega. O empacotamento deve facilitar a remoção dos itens. Neste caso, é preciso combinar um problema de empacotamento com um problema de roteamento.

Existem ainda situações em que é preciso empacotar cada item no momento em que os dados sobre o item estiverem disponíveis, sem que se conheça de antemão todos os itens que deverão ser empacotados. Ademais, uma vez empacotado, não é permitido mudar a posição do item dentro do padrão. Os problemas que têm estas características são chamados de problemas de empacotamento *on-line*.

3.2.1.6 Restrições dos padrões

Existem diversas restrições que podem ser impostas aos padrões em função de certas peculiaridades dos processos de produção. Destacamos quatro grupos de restrições:

- Distâncias mínimas ou máximas entre os itens ou entre os cortes são frequentemente importantes, por exemplo, no corte de vidro ou no carregamento de contêineres.
- A orientação dos itens em relação ao objeto tem que ser levada em consideração, por exemplo, no empacotamento de itens frágeis.
- Em certas situações a quantidade de vezes que um item pode aparecer no padrão ou o número de figuras distintas no padrão deve ser limitado.
- O tipo e o número de cortes permitidos também podem ser restritos. Por exemplo, quando os objetos e os itens têm formas retangulares, os padrões podem ser classificados como *ortogonais*, se todos os cortes são perpendiculares a algum dos lados do objeto, ou *não-ortogonais*, caso contrário. No caso de corte guilhotinado pode haver restrição com respeito ao número de estágios de cortes, bem como no número de cortes paralelos por estágio. Cada mudança na direção dos cortes determina um novo estágio de corte.

3.2.1.7 Restrições de alocação

O processo de alocação dos itens dentro dos objetos, gerando os padrões, pode apresentar diversas restrições, muitas delas estreitamente relacionadas com a disponibilidade dos itens e objetos, característica discutida anteriormente. Algumas restrições comuns são:

- O tipo de alocação determina, tanto para os objetos quanto para os itens, se todos ou apenas uma parte deles deve ser utilizada. Os tipos de alocação mais comuns são: (a) utilizar todos os objetos e apenas uma parte dos itens; (b) utilizar uma parte dos objetos e todos os itens. O problema das mochilas múltiplas é um exemplo do primeiro tipo de alocação. Já o problema de corte de estoque bidimensional com demandas é um exemplo do segundo tipo de alocação.

- O número de estágios de alocação define se os itens devem ser alocados simultaneamente ou em estágios sucessivos. No segundo caso as figuras residuais de um estágio tornam-se os objetos dos estágios seguintes.

- A alocação dos itens dentro dos objetos pode ser de natureza estática ou dinâmica. Na alocação estática os itens são alocados sem possibilidade de remanejamento e normalmente não são conhecidos os próximos itens ou objetos a serem utilizados, característica básica dos processos de corte e empacotamento ditos *on-line*. Em contrapartida, nos processos *off-line* a alocação é dinâmica, e existe a possibilidade de realocação dos itens dentro dos objetos.

3.2.1.8 Objetivos

Nem sempre é possível definir exatamente se uma característica é geométrica ou combinatória. Algumas incluem os dois aspectos, outras nenhum. Os objetivos dos problemas de corte e empacotamento frequentemente têm aspectos geométricos, outras vezes têm aspectos combinatórios. Ademais, os objetivos podem estar relacionados com os objetos, com os itens, com os padrões ou com os processos de alocação. O objetivo de um problema de corte ou empacotamento pode ser definido como um critério a ser maximizado ou minimizado. Podemos citar como exemplos de objetivos:

- Minimizar a quantidade de objetos utilizados;

- Minimizar a soma dos custos dos objetos utilizados. Neste caso deve haver um custo associado a cada objeto;

- Minimizar o desperdício nos padrões;

- Maximizar o valor dos itens produzidos. Os itens podem ter valores arbitrários ou valores proporcionais ao seu tamanho.

A lista anterior, embora não exaustiva, relaciona os objetivos dos principais problemas encontrados na literatura. Existem também problemas onde vários objetivos têm que ser considerados [Wäs90].

3.2.1.9 Natureza da informação e variabilidade

Os dados dos problemas podem ser de natureza determinística, estocástica ou incerta. Além disso, os dados podem ser valores estritos ou intervalos de valores. A inexatidão nos processos de medição é um dos principais fatores que ocasiona variabilidade nos dados do problema.

Por exemplo, na indústria de construção civil, as barras de aço comercializadas pelos distribuidores deveriam possuir 1200 centímetros de comprimento [1]. No entanto, é comum as barras possuírem comprimentos levemente variados. Além disso, o peso das barras de aço de mesmo comprimento e da mesma categoria, que deveria ser igual, também apresenta variação. Esta imprecisão tem origem na indústria siderúrgica e provoca um efeito chamado de *desbitolamento* que se não for mantido dentro de limites aceitáveis pode comprometer a qualidade das estruturas de concreto armado.

Estes fatores podem levar a definição de variantes com características especiais, as quais devem ser levadas em consideração pelos métodos empregados para resolver tais variantes.

Apresentamos na Tabela 3.1 um resumo das principais características dos objetos, itens, padrões e processos de alocação dos problemas de corte e empacotamento. Observe que algumas características têm natureza puramente geométrica, outras têm natureza puramente combinatória e existem ainda outras que combinam aspectos geométricos e combinatórios.

3.2.2 Classificação de Dyckhoff

Certamente as características abordadas na subseção anterior não incluem todas as possíveis propriedades dos problemas de corte e empacotamento. Além disso, várias características citadas se sobrepõem. Por exemplo, a disponibilidade e o tipo de alocação

[1] Esta medida padrão é adequada para os caminhões que fazem o transporte das barras de aço. Além disso, o fato do número 1200 possuir um grande número de divisores ajuda a diminuir o desperdício no padrões, visto que, na prática, os itens especificados nas plantas de ferragem frequentemente têm comprimentos que são divisores de 1200.

Características dos...	Características geométricas	Características combinatórias	Outras características
Objetos e itens	Dimensionalidade Formato das figuras	Medidas de quantidade Sortimento Disponibilidade	Objetivos Status da informação Variabilidade
Padrões	Dimensionalidade Restrições dos padrões (figuras; tipos de corte; distâncias; orientação...)	Restrições dos padrões (número de cortes; tipo, número e combinação das figuras)	Objetivos Status da Informação Variabilidade
Processos de alocação	-	Restrições no número de estágios, ordem ou frequência dos padrões	Objetivos Status da informação Variabilidade

Tabela 3.1: Sistematização das principais características.

são características estreitamente relacionadas. Mesmo assim, baseado em algumas das características vistas anteriormente, Dyckhoff propôs um esquema que permite classificar os diversos tipos de problemas de corte e empacotamento de maneira consistente e sistemática.

Tal esquema define classes de problemas combinando-se os tipos principais de quatro características básicas. As quatro características assim como seus tipos principais, denotados por seus respectivos símbolos, são:

1. Dimensionalidade
   (1) Unidimensional
   (2) Bidimensional
   (3) Tridimensional
   (N) N-dimensional ($N > 3$)

2. Tipo de alocação
   (B) Todos os objetos e uma parte dos itens
   (V) Uma parte dos objetos e todos os itens

3. Sortimento dos objetos
   (O) Um objeto
   (I) Objetos de figuras idênticas
   (D) Objetos de diferentes figuras

4. Sortimento dos itens
   (F) Poucos itens (ou figuras diferentes)
   (M) Muitos itens de muitas figuras diferentes
   (R) Muitos itens de relativamente poucas figuras distintas
   (C) Figuras congruentes (mesma forma e tamanho)

Cada classe de problema de corte e empacotamento é definida através de uma quádrupla $\alpha/\beta/\gamma/\delta$, onde α é a dimensionalidade, β o tipo de alocação, γ o sortimento dos objetos e δ o sortimento dos itens. Por exemplo, a quádrupla 1/V/I/R denota a classe de problemas unidimensionais onde todos os itens, de relativamente poucas figuras distintas, devem ser alocados dentro de objetos de mesmo tamanho.

É possível obter classes mais abrangentes deixando de especificar parte dos símbolos da quádrupla, indicando que as características não especificadas podem ser de qualquer um dos tipos principais. Por exemplo, a classe 1/B/O/ inclui o problema da mochila inteira, onde o sortimento dos itens pode ser de qualquer tipo.

Na Tabela 3.2 listamos alguns problemas de corte e empacotamento abordados na literatura, indicando a classe a que pertencem, segundo a classificação de Dyckhoff. Pela tabela podemos perceber que dentro de uma mesma classe podemos encontrar diversos problemas que se diferenciam por outras características além das quatro principais adotadas na classificação. Por exemplo, o problema de alocação de memória, o problema de empacotamento de *bins*, o problema da linha de montagem e o problema de alocação de tarefas em multiprocessador pertencem todos à mesma classe (1/V/I/M), mas possuem características diferentes não incluídas na classificação aqui definida.

3.3 Problemas de Corte e Empacotamento Bidimensional

Conforme mencionamos na introdução, nossa pesquisa concentrou-se em problemas de corte e empacotamento bidimensional. O objetivo desta seção é definir formalmente as variantes destes problemas que abordaremos nos capítulos seguintes. Inicialmente, vejamos uma definição geral para estes problemas, que servirá de base para a definição das variantes.

Os problemas de corte e empacotamento bidimensional consistem basicamente em: dados a largura L e a altura A de objetos retangulares, genericamente denominados de

Problema	Classe
Mochila inteira	1/B/O/
Mochila compartimentada	1/B/D/
Mochila multidimensional	/B/O/
Mochilas múltiplas	1/B/D/
Carregamento de páletes do produtor	2/B/O/C
Carregamento de contêineres	3/V/I/ ou 3/B/O/
Empacotamento de *bins*	1/V/I/M
Empacotamento em faixa	2/V/O/M
Empacotamento em altura	3/V/O/M
Empacotamento de quadrados em quadrados	2/V/I/M
Empacotamento de cubos N-dimensional	N/V/I/M
Corte de estoque unidimensional	1/V/I/R
Corte de estoque bidimensional	2/V/I/R
Corte de estoque bidimensional binário	2/V/I/M
Corte de estoque tridimensional	3/V/I/R
Corte de guilhotina bidimensional com demanda	2/V/I/R
Corte de guilhotina bidimensional binário	2/V/I/M
Corte de guilhotina bidimensional com valor	2/B/O/R
Linha de montagem	1/V/I/M
Alocação de tarefas em multiprocessador	1/V/I/M
Alocação de memória	1/V/I/M
Planejamento de investimentos multiperiódicos	N/B/O/

Tabela 3.2: Alguns problemas de corte e empacotamento e sua classificação.

placas, e uma lista de m itens retangulares, cada item i com largura $l_i \leq L$ e altura $a_i \leq A$, queremos determinar "a melhor maneira" de cortar placas de forma a produzir itens da lista (ou equivalentemente, empacotar itens nas placas). Dependendo das situações em que estes problemas surgem, o termo "a melhor maneira" pode ter significados bem diferentes, conforme veremos mais adiante.

Chamamos a atenção do leitor para o fato de que *cortar* uma placa de forma a produzir itens de uma lista equivale a *empacotar* itens da lista dentro da placa. Dessa forma, para fins de formulação e resolução, podemos considerar problemas de corte de estoque como sendo análogos a problemas de empacotamento, embora em situações práticas existam diferenças significativas entre os processos de corte de estoque e os processos de empaco-

tamento. Por exemplo, o problema de corte de estoque bidimensional com demandas, que definiremos mais adiante, é análogo ao problema de empacotamento bidimensional com demandas.

Sendo assim, nos capítulos seguintes, na descrição de alguns algoritmos para problemas de corte, quando conveniente usamos o termo empacotar, sem com isso mudar o caráter do problema. Nesse contexto, o termo empacotar deve ser entendido como fazer o corte correspondente ao empacotamento em questão.

Consideraremos a largura como a dimensão horizontal (eixo x) e a altura como a dimensão vertical (eixo y). Denotaremos as dimensões de uma placa (ou de um item) por um par ordenado (a, b), onde a e b representam a largura e a altura da placa (ou do item), respectivamente.

Supomos aqui que os cortes são infinitamente finos, ou seja, têm largura zero. Apesar de isto não ocorrer na prática, esta suposição não é na verdade uma restrição, pois se num processo de corte do mundo real todos os cortes têm largura δ, basta somar δ a L, A, $l_1, \ldots, l_m, a_1, \ldots, a_m$, e resolver esta nova instância (agora supondo que todos os cortes têm largura zero) [Bea85a].

A seguir, discutiremos algumas especificidades dos problemas de corte e empacotamento bidimensional e definiremos as variantes que serão discutidas nos próximos capítulos.

3.3.1 Atribuindo Valores

Podemos associar a cada item i um valor v_i $(i = 1, \ldots, m)$. Chamaremos de *problema de corte de estoque bidimensional com valor* (PCEV_2) a variante onde cada item i possui valor v_i $(i = 1, \ldots, m)$ e desejamos determinar como cortar uma única placa de modo a maximizar a soma dos valores dos itens produzidos.

Se desejarmos simplesmente encontrar um padrão que minimize a área não utilizada da placa, que chamaremos de *sobras*, basta atribuir a cada item i valor $l_i.a_i$, ou seja, valor equivalente à sua área. Dessa forma, maximizar a soma dos valores dos itens produzidos equivale a minimizar as sobras.

Dado um padrão, as regiões da placa que não são ocupadas por nenhum item (sobras) possuem valor zero. Na Figura 3.1 (e nas demais figuras), estas regiões são representadas por polígonos escuros.

3.3.2 Introduzindo Demandas

Em muitas situações precisamos produzir uma quantidade pré-determinada de cópias de cada item. Introduzimos esta exigência na definição do problema associando a cada item i uma demanda não-nula $d_i \in \mathbb{N}$ $(i = 1, \ldots, m)$.

Chamaremos de *problema de corte de estoque bidimensional com demandas* (PCED_2) a variante onde cada item i possui demanda d_i $(i = 1, \ldots, m)$ e desejamos determinar como produzir d_i unidades de cada item i utilizando o menor número de placas possível. Observe que no PCED_2 não tem importância atribuir valor aos itens visto que em qualquer solução temos que produzir exatamente d_i unidades de cada item i.

A variante do PCED_2 onde todos os itens têm demanda igual a 1 será chamada de *problema de corte de estoque bidimensional binário* (PCEB_2).

3.3.3 Placas e Itens Quadrados

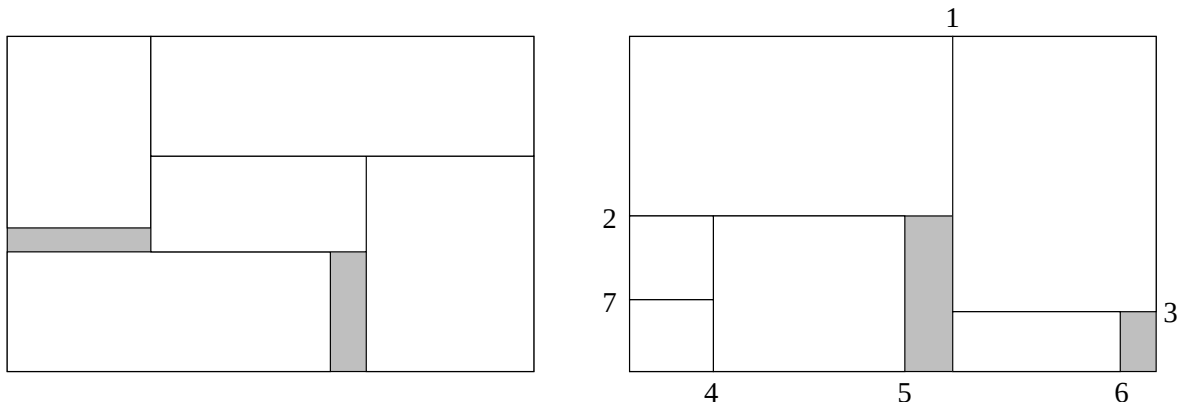
É interessante estudar a variante do PCED_2 em que todas as placas e itens são quadrados. Chamaremos tal variante de *problema de empacotamento de quadrados em quadrados com demandas* (PEQD). Se a demanda de todos os itens for igual a 1, chamaremos de *problema de empacotamento de quadrados em quadrados binário* (PEQB).

3.3.4 Cortes de Guilhotina

Uma restrição bastante comum nos processos que envolvem corte de placas é exigir que todos os cortes sejam ortogonais a um dos lados da placa. Chamamos tal restrição de *restrição de ortogonalidade*. Em alguns casos é requerido ainda que os cortes se estendam em linha reta desde um lado até o lado oposto da placa. Tais cortes são chamados de *cortes de guilhotina*.

Dizemos que um padrão é *guilhotinável* se pode ser obtido com uma sequência de cortes de guilhotina aplicados à placa original e às subplacas que são obtidas após cada corte. Na Figura 3.1 temos, à esquerda, um padrão não-guilhotinável e, à direita, um padrão guilhotinável (a numeração indica a ordem em que os cortes de guilhotina podem ser feitos).

Dado um padrão guilhotinável, chamamos de *estágio de corte* (ou simplesmente *estágio*) uma sequência maximal de cortes consecutivos, todos na mesma direção. Por exemplo, no padrão da Figura 3.1b, no primeiro estágio é feito o corte 1, no segundo estágio são



(a) Padrão não-guilhotinável (b) Padrão guilhotinável

Figura 3.1: Padrões de corte

feitos os cortes 2 e 3, no estágio seguinte os cortes 4, 5 e 6, e no último estágio é feito o corte 7.

Em muitos processos de corte do mundo real a quantidade de estágios necessários para se produzir um padrão pode ser limitada. Diversos autores investigaram o caso em que o número de estágios está limitado a dois ou três [RST95, Hif01]. Esta restrição diminui a quantidade de padrões viáveis e portanto facilita a resolução do problema. Em nosso trabalho, vamos supor que a quantidade de estágios é ilimitada.

Nos problemas de corte de guilhotina também podemos atribuir valores e demandas aos itens. Chamaremos de *problema de corte de guilhotina bidimensional com valor* (PCGV_2) a variante do PCEV_2 onde o padrão produzido tem que ser guilhotinável. Como veremos no Capítulo 6, o PCGV_2 possui propriedades que nos permitem aplicar a técnica de programação dinâmica para resolvê-lo.

Chamaremos de *problema de corte de guilhotina bidimensional com demandas* (PCGD_2) a variante do PCED_2 onde todos os padrões têm que ser guilhotináveis. No capítulo 7 veremos que o PCGD_2 pode ser formulado como um PLI. Mostraremos como resolver a relaxação linear desse PLI através do método de geração de colunas e a partir daí encontrar uma solução para o PCGD_2. Chamaremos de *problema de corte de guilhotina bidimensional binário* (PCGB_2) a variante do PCGD_2 onde todos os todos os itens têm demanda igual a 1.

3.3.5 Rotações Ortogonais

Em diversas situações práticas é permitido que os itens sofram rotações ortogonais, ou seja, de 90 graus. Por brevidade, chamaremos rotações ortogonais simplesmente de *rotações*. Em outras situações, como por exemplo, no corte de tecido estampado, de vidro trabalhado, etc, rotações não são permitidas.

É possível transformar uma instância do PCEV_2 onde rotações são permitidas numa instância onde rotações não são permitidas da seguinte maneira: para cada item i, de largura l_i, altura a_i e valor v_i, inserimos um item de largura a_i, altura l_i e valor v_i, desde que $l_i \neq a_i$, $l_i \leq A$ e $a_i \leq L$.

De forma análoga, podemos transformar uma instância do PCGV_2 onde rotações são permitidas numa instância onde rotações não são permitidas. Sendo assim, vamos supor que em todas as instâncias do PCEV_2 e do PCGV_2 rotações não são permitidas (se necessário, aplicamos a transformação aqui explicada).

Nos demais problemas que abordaremos, vamos supor que rotações não são permitidas, a menos que seja explicamente indicado o contrário.

3.3.6 Faixa de Altura Ilimitada

Uma variante do PCED_2 que tem sido largamente estudada é chamada de *problema de empacotamento em faixa* (PEF). Neste caso, têm-se apenas uma placa de largura L e altura ilimitada. Chamaremos esta placa de *faixa*. Desejamos empacotar os itens na faixa de forma que a altura utilizada seja a menor possível.

Se a demanda de todos os itens for igual a 1, chamaremos a variante de *problema de empacotamento em faixa binário* (PEFB).

3.3.7 Placas de Dimensões Variadas

Em algumas situações, as placas podem ter dimensões variadas. Neste caso, faz sentido atribuir um custo a cada tipo de placa. Tal problema, que chamaremos de *problema de corte de guilhotina bidimensional com demandas e placas de dimensões variadas* (PCGD_2V), consiste em: dada uma lista de k tipos de placas, onde placa do tipo j tem largura L_j, altura A_j e custo C_j, e uma lista de m itens, cada item i com largura l_i, altura a_i e demanda d_i, determinar como produzir d_i unidades de cada item i, utilizando apenas

cortes de guilhotina, de forma que a soma dos custos das placas utilizadas seja a menor possível.

Ao contrário de outras variantes, existe bem pouca literatura a respeito do $PCGD_2V$. Podemos citar, entretanto, a heurística proposta por Yanasse, Zinober e Harris [YZH91] que resolve instâncias do $PCGD_2V$ onde o objetivo é minimizar as sobras nas placas utilizadas[2]. Este objetivo é equivalente a minimizar a soma dos custos das placas utilizadas se considerarmos que o custo de cada placa é linearmente proporcional à sua área.

Existem ainda outras variantes extensamente estudadas na literatura, mas que não abordaremos neste trabalho. Por exemplo, Christofides e Whitlock [CW77] propuseram associar a cada item i um inteiro b_i $(i = 1, \ldots, m)$, que representa o número máximo de vezes que o item i pode aparecer num padrão. Em geral, tal restrição é arbitrária. Neste caso, seu objetivo é diminuir a quantidade de padrões viáveis, numa tentativa de facilitar a resolução do problema. Esta variante é conhecida como *problema de corte de estoque bidimensional restrito*. Diversos algoritmos foram propostos para este problema, como por exemplo [CW77, Wan83, OF90, DAD95].

Nos capítulos seguintes discutiremos algoritmos para resolver algumas das variantes dos problemas de corte e empacotamento bidimensional definidas nesta seção. Na Tabela 3.3 listamos estas variantes. Além disso, indicamos a sua caracterização com respeito aos aspectos discutidos nesta seção e a classe a que pertencem estas variantes, segundo a classificação de Dyckhoff.

Característica	$PCEB_2$	PEQD	$PCGV_2$	$PCGD_2$	PEFB	$PCGD_2V$
Itens com valores			Sim			
Itens com demandas	Sim	Sim		Sim	Sim	Sim
Demandas binárias	Sim				Sim	
Figuras quadradas		Sim				
Cortes de guilhotina			Sim	Sim		Sim
Rotações ortogonais	Sim			Sim	Sim	Sim
Altura ilimitada					Sim	
Placas variadas						Sim
Classe	2/V/I/M	2/V/I/R	2/B/O/R	2/V/I/R	2/V/O/M	2/V/D/R

Tabela 3.3: Exemplos de problemas de corte e empacotamento bidimensional.

[2]De fato, o problema estudado por estes autores inclui duas restrições adicionais: a quantidade de cortes de guilhotina no primeiro estágio é limitada e as bordas da placa devem ser aparadas, pois durante o processo de corte elas são usadas para fixação da placa.

Capítulo 4

Algoritmos de Aproximação para Problemas de Empacotamento

4.1 Introdução

Nas últimas quatro décadas, um grande número de algoritmos têm sido propostos para se resolver diversas variantes dos problemas de corte e empacotamento, sendo vasta a literatura a este respeito. Diversos artigos de revisão bibliográfica sobre o assunto têm sido escritos nos últimos anos. Recomendamos ao leitor interessado os artigos de Lodi, Martello e Monaci [LMM02], Friedman [Fri98], Coffman, Garey e Johnson [CGJ97], Galambos e Woeginger [GW95], Cheng, Feiring e Cheng [CFC94], Dyckhoff e Finke [DF92], Sweeney e Paternoster [SP92] e Ram [Ram92].

Não é nosso propósito discorrer longamente sobre as dezenas de algoritmos propostos para os problemas de corte e empacotamento, mas sim nos concentrar naqueles algoritmos de aproximação que estão de alguma forma relacionados com os novos algoritmos que vamos propor nos capítulos seguintes. Alguns dos algoritmos que discutiremos neste capítulo serão usados como sub-rotina, outros servem como inspiração e alguns serão usados para fins de comparação com os nossos algoritmos.

Neste capítulo, discutiremos diversos algoritmos de aproximação, sendo que o nosso interesse maior recai sobre o algoritmo *Hybrid First Fit* (HFF), que será usado como sub-rotina em um dos algoritmos que propomos no Capítulo 7. Como o HFF utiliza como sub-rotina os algoritmos *First Fit Decreasing* (FFD) e *First Fit Decreasing Height* (FFDH), estes dois últimos algoritmos também serão apresentados. O FFD é um refinamento do algoritmo *First Fit* (FF) e o FFDH é um refinamento do algoritmo *Next Fit Decreasing Height* (NFDH). Sendo assim, discutiremos também os algoritmos FF e

NFDH. Finalmente, o algoritmo FF é baseado no algoritmo *Next Fit*, que também será explicado. Apresentaremos estes seis algoritmos na ordem em que foram originalmente propostos, começando pelo FF, o mais simples, até chegar ao HFF, o mais complexo.

Citaremos ainda outros algoritmos e esquemas de aproximação correlatos, que podem ser de interesse do leitor, fornecendo suas respectivas razões de aproximação. Os detalhes destes algoritmos e esquemas podem ser obtidos nas referências bibliográficas apresentadas.

4.2 Algoritmos para o PEBB

Conforme mencionamos anteriormente, estamos particularmente interessados no algoritmo HFF. Antes de apresentá-lo, porém, é conveniente abordar outros algoritmos nos quais o HFF está baseado. Precisamos discutir inicialmente alguns algoritmos de aproximação para o *problema de empacotamento de bins binário* (PEBB).

Primeiramente, vejamos a definição do PEBB: dada uma quantidade ilimitada de objetos unidimensionais de comprimento L, usualmente denominados de *bins*, e uma lista de m itens unidimensionais, onde cada item i possui comprimento $l_i \leq L$ e demanda exatamente igual a 1, desejamos determinar como empacotar os itens nos *bins* utilizando o menor número possível de *bins*. Veremos a seguir três algoritmos de aproximação para o PEBB.

4.2.1 O Algoritmo *Next Fit*

Um dos mais simples algoritmos de aproximação para o PEBB citados na literatura é o *Next Fit* (NF). Este algoritmo foi analisado (quanto ao seu desempenho assintótico) por David Johnson [Joh73] em sua famosa tese de doutorado, juntamente com os algoritmos FF e FFD, que serão apresentados mais adiante.

Para tornar mais clara a explicação dos algoritmos NF, FF e FFD, vamos considerar que os *bins* estão dispostos na direção do eixo x (horizontal), sendo o canto esquerdo do *bin* correspondente à posição 0 (zero) do eixo x. Consideraremos os *bins* indexados a partir de 1 e chamaremos o *bin* j de B_j. Ademais, representaremos por c_j o comprimento utilizado em B_j, ou seja, a posição mais à direita ocupada por algum item empacotado em B_j. Obviamente, antes de iniciar o processo de empacotamento, temos que $c_j = 0$, para todo B_j.

Iniciamos o algoritmo empacotando o item 1 no canto esquerdo de B_1. Suponha que o item i é o próximo item a ser empacotado e j é o maior índice tal que $c_j > 0$. Se $l_i + c_j \leq L$, então o item i cabe em B_j e portanto empacotamos o item i na posição c_j de B_j [1] e atualizamos o comprimento utilizado em B_j fazendo $c_j = c_j + l_i$. Observe que c_j é a posição mais à esquerda dentro do *bin* na qual é possível empacotar o item i sem que haja sobreposição de itens. Se $l_i + c_j > L$, então o item i não cabe em B_j. Neste caso, passamos a utilizar o próximo *bin*; fazemos $j = j + 1$, empacotamos o item i no canto esquerdo de B_j e atualizamos c_j fazendo $c_j = l_i$. Repetimos este procedimento até que todos os itens tenham sido empacotados. Apresentamos a seguir uma descrição formal do algoritmo.

Algoritmo 4.1 Next Fit (NF)

Entrada: Uma instância $I = (L, l)$ do PEBB, onde $l = (l_1, \ldots, l_m)$.

Saída: Uma solução para I.

Faça $j = 1$ e $c_1 = 0$.
Para $i = 1$ até m
  Se $l_i + c_j > L$ faça $j = j + 1$ e $c_j = 0$. `/* É preciso usar um novo bin */`
  Empacote o item i na posição c_j de B_j e faça $c_j = c_j + l_i$.
Devolva o empacotamento.

Claramente, este algoritmo pode ser implementado de forma a ter complexidade de tempo linear. Ademais, Johnson [Joh73] mostrou que o NF possui razão de aproximação assintótica igual a 2 e que esta razão é justa. Mostrou ainda que a razão de aproximação absoluta do NF também é igual a 2.

4.2.2 O Algoritmo *First Fit*

É possível obter um algoritmo com uma razão de aproximação melhor introduzindo uma pequena modificação no NF. Ao empacotar o item i, procuramos o menor índice j tal que $c_j > 0$ e $c_j + l_i \leq L$, ou seja, determinamos o *bin* de menor índice que está em uso e no qual cabe o item i. Se existir tal j, empacotamos o item i na posição c_j de B_j e fazemos $c_j = c_j + l_i$. Caso contrário, passamos a utilizar um novo *bin* e empacotamos o item i no canto esquerdo deste novo *bin*. Este algoritmo é chamado de *First Fit* (FF) e seus detalhes são dados a seguir.

[1] Empacotar o item i na posição c_j de B_j significa posicionar o item i dentro de B_j de forma que a extremidade esquerda do item esteja na posição c_j de B_j.

Algoritmo 4.2 First Fit (FF)

Entrada: Uma instância $I = (L, l)$ do PEBB, onde $l = (l_1, \ldots, l_m)$.

Saída: Uma solução para I.

Faça $j = 1$ e $c_1 = 0$.
Para $i = 1$ até m
  Seja $k = min(\{h \mid 1 \leq h \leq j,\ l_i + c_h \leq L\} \cup \{j+1\})$.
  Se $k = j + 1$ faça $j = j + 1$ e $c_j = 0$. `/* É preciso usar um novo bin */`
  Empacote o item i na posição c_j de B_j e faça $c_j = c_j + l_i$.
Devolva o empacotamento.

A diferença fundamental entre os algoritmos NF e FF é que no primeiro tentamos empacotar cada item somente no último *bin* que passou a ser utilizado. Se isto não for possível, passamos a utilizar um novo *bin*, ignorando possíveis espaços restantes nos *bins* anteriores. Por outro lado, no FF, cada item é empacotado no *bin* de menor índice no qual ele caiba. Com isto, espaços remanescentes em todos os *bins* que já estão em uso podem vir a ser aproveitados.

Note que precisaremos de no máximo m *bins* para empacotar m itens. Sendo assim, encontrar o *bin* de menor índice no qual cabe o item i, pode ser feito em tempo polinomial, portanto o FF também é um algoritmo polinomial.

Pode-se mostrar ainda que $\mathrm{FF}(I) \leq \frac{17}{10}\,\mathrm{OPT}(I) + 1$ e que a razão de aproximação assintótica $\frac{17}{10}$ do FF também é justa [JDU+74, GGJY76, GGJ78]. Ademais, Simchi-Levi [SL94] mostrou que a razão de aproximação absoluta do FF é 1,75.

4.2.3 O Algoritmo *First Fit Decreasing*

Podemos obter uma melhora significativa na razão de aproximação assintótica do FF através de uma modificação bastante simples. A alteração consiste em primeiramente ordenar os elementos da lista de itens em ordem decrescente de comprimento. O restante do algoritmo permanece inalterado. Tal algoritmo é conhecido como *First Fit Decreasing* (FFD).

Claramente, o FFD é um algoritmo polinomial. Na verdade, o FFD pode ser implementado de forma a ter complexidade de tempo $\mathcal{O}(m \log m)$, que é a mesma complexidade do algoritmo FF. Sobre a razão de aproximação do FFD, diversos pesquisadores demonstraram o seguinte teorema [Joh73, JDU+74, Bak85].

Algoritmo 4.3 First Fit Decreasing (FFD)

Entrada: Uma instância $I = (L, l)$ do PEBB, onde $l = (l_1, \ldots, l_m)$.

Saída: Uma solução para I.

Ordene os itens em ordem decrescente de comprimento, obtendo $l_1 \geq \ldots \geq l_m$.
Faça $j = 1$ e $c_1 = 0$.
Para $i = 1$ até m
  Seja $k = min(\{h \mid 1 \leq h \leq j,\ l_i + c_h \leq L\} \cup \{j+1\})$.
  Se $k = j + 1$ faça $j = j + 1$ e $c_j = 0$. `/* É preciso usar um novo bin */`
  Empacote o item i na posição c_j de B_j e faça $c_j = c_j + l_i$.
Devolva o empacotamento.

Teorema 4.2.1 *Para toda instância I do PEBB temos que*

$$\mathrm{FFD}(I) \leq \frac{11}{9}\ \mathrm{OPT}(I) + 4.$$

Ademais, a razão de aproximação assintótica $\frac{11}{9}$ do FFD é justa.

Em 1991, Yue [Yue91] mostrou que a constante aditiva do Teorema 4.2.1 pode ser substituída por 1. Além disso, segundo Bramel, Rhee e Simchi-Levi [BRSL97], o FFD apresenta desempenho empírico no caso médio de 1,02. Trata-se portanto de um algoritmo bastante eficiente na prática, tanto em termos de tempo quanto no que se refere à qualidade das soluções encontradas.

Apesar de ser empiricamente bom no caso médio, Simchi-Levi [SL94] mostrou que a razão de aproximação absoluta do FFD é 1,5 e que esta razão é justa. Ademais, sabe-se que não existe algoritmo de aproximação para o PEBB com razão de aproximação absoluta menor que 1,5 [GJ79], a menos que $\mathcal{P} = \mathcal{NP}$.

Foi também observado que o FFD encontra dificuldade, no que diz respeito à qualidade da solução, em instâncias pequenas ou naquelas em que os itens têm comprimentos próximos de $\frac{1}{3}L$, $\frac{1}{4}L$, $\frac{1}{5}L$, ... (*cf.* Schwerin e Wäscher [SW97]) ou ainda nas instâncias que Falkenauer [Fal96] chamou de *triplets*.

Vale observar que no FFD precisamos conhecer a priori todos os itens que devem ser empacotados (para que seja possível ordená-los), para então começar a empacotá-los. Já no NF e no FF isto não é necessário, podemos empacotar os itens à medida que estes são dados. Dizemos que o FFD é um algoritmo *off-line*, enquanto que o NF e o FF são algoritmos *on-line*.

Esta propriedade é relevante em certas aplicações práticas nas quais temos que empacotar os itens no instante em que sua descrição é fornecida, sem conhecer os demais itens que deverão ser empacotados. Em tais situações, precisamos utilizar algoritmos *on-line*.

Existem algoritmos *on-line* para o PEBB com razão de aproximação assintótica melhor que o NF e o FF. Por exemplo, o algoritmo proposto por Richey [Ric91] possui razão de aproximação assintótica igual a 1,588. Por outro lado, Van Vliet [Vli92] provou que não existe algoritmo *on-line* para o PEBB com razão de aproximação assintótica menor que 1,5401.

Foram propostos vários outros algoritmos de aproximação para o PEBB, tais como o *Best Fit* (BF), o *Next Fit Decreasing* (NFD), o *Best Fit Decreasing* (BFD) e o *Modified First Fit Decreasing* (MFFD) [JDU$^+$74, Joh74, KSS75, Joh73]. As razões de aproximação assintótica dos algoritmos BF, NFD, BFD e MFFD são $\frac{17}{10}$, 1,691..., $\frac{11}{9}$ e $\frac{71}{60}$, respectivamente. Ademais, estas razões são justas. Destes algoritmos, apenas o BF é *on-line*. Além disso, Simchi-Levi [SL94] mostrou que o BF possui razão de aproximação absoluta igual a 1,75.

Sabe-se ainda que o PEBB admite esquemas de aproximação. Fernandez de la Vega e Lueker [FL81] desenvolveram um FPAAS para o PEBB. Estes autores mostraram que para cada $\epsilon > 0$, existe um algoritmo FL_ϵ que é polinomial em ϵ e no tamanho da instância I e tal que $\mathrm{FL}_\epsilon(I) \leq (1+\epsilon)\ \mathrm{OPT}(I) + (\frac{1}{\epsilon})^2$. Recentemente, Coffman, Garey e Johnson [CGJ97] provaram que a constante aditiva pode ser trocada para $\frac{1}{\epsilon}$ e que um algoritmo VL_ϵ pode ser implementado de forma a ter complexidade de tempo $\mathcal{O}(\epsilon^{-c} m \log m)$, para uma constante positiva c. No entanto, este FPAAS possui significado principalmente teórico, devido ao fato de que a constante aditiva é demasiadamente grande para ϵ bem pequeno.

Karmarkar e Karp [KK82] propuseram o algoritmo KK, que utiliza programação linear para resolver o PEBB. Eles provaram que $\mathrm{KK}(I) \leq\ \mathrm{OPT}(I) + \mathcal{O}(\log^2\ \mathrm{OPT}(I))$ e que este algoritmo tem complexidade de tempo $\mathcal{O}(m^8 \log^3 m)$.

4.3 Algoritmos para o PEFB

Veremos agora dois algoritmos de aproximação para o problema de empacotamento em faixa binário (PEFB). Vale lembrar que no PEFB tem-se apenas uma faixa de largura L e altura ilimitada e uma lista de m itens retangulares, cada item i com largura l_i, altura a_i e demanda igual a 1. Desejamos empacotar todos os itens na faixa de modo que a altura utilizada da faixa seja a menor possível. Além disso, não é permitido que os itens

sofram rotações.

O primeiro algoritmo que discutiremos, conhecido como *Next Fit Decreasing Height* (NFDH) combina a estratégia do algoritmo NF com a idéia de ordenação presente no FFD. O segundo algoritmo, chamado de *First Fit Decreasing Height* (FFDH), é uma adaptação do FFD para o PEFB. Estes dois algoritmos foram propostos por Coffman, Garey, Johnson e Tarjan [CGJT80].

Antes de descrever estes algoritmos, vejamos como representar um empacotamento no caso do PEFB. Utilizaremos um sistema de coordenadas bidimensional e denotaremos uma posição neste sistema de coordenadas através de um par ordenado (a, b), onde a representa a posição no eixo x e b representa a posição no eixo y.

Vamos convencionar que a faixa tem largura L ao longo do eixo x (horizontal) e altura ilimitada ao longo do eixo y (vertical). Cada item i tem largura l_i ao longo do eixo x e altura a_i ao longo do eixo y $(i = 1, \ldots, m)$. Convencionamos que o canto inferior esquerdo da faixa está na posição $(0, 0)$ do nosso sistema de coordenadas.

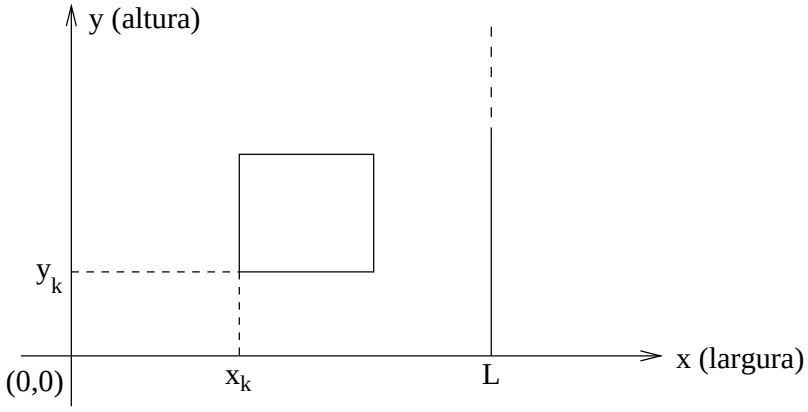


Figura 4.1: Sistema de coordenadas

Desse modo, para especificar a posição de um item k na faixa, basta especificar a posição do canto inferior esquerdo desse item em relação ao sistema de coordenadas. Neste caso, por causa da restrição de ortogonalidade (veja Subseção 3.3.4), se (x_k, y_k) denota um tal canto esquerdo e as dimensões deste item são (l_i, a_i), a região ocupada por ele é dada por

$$\mathcal{R}_k = \{(x, y) \mid x_k \leq x \leq x_k + l_i \text{ e } y_k \leq y \leq y_k + a_i\}.$$

Com essas convenções, definimos um empacotamento para o PEFB como sendo uma

coleção de itens $\wp$ que satisfaz as seguintes propriedades:

(a) Para todo item i em $\wp$, temos que $0 \leq x_i \leq L - l_i$ e $y_i \geq 0$;

(b) Para todo par de itens distintos i, j em $\wp$, temos que $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$.

Passamos agora a descrever os algoritmos NDFH e FFDH.

4.3.1 O Algoritmo *Next Fit Decreasing Height*

No NFDH, os itens são empacotados em níveis (camadas) horizontais dentro da faixa, da esquerda para a direita. Vamos indexar os níveis, a partir de 1, de acordo com a ordem em que forem criados (veja Figura 4.2). Denotaremos por t_j a posição correspondente ao topo (posição mais alta) do nível j, e por w_j a largura utilizada no nível j, ou seja, a posição mais à direita ocupada por algum item empacotado neste nível. A altura de um nível j é a altura do item mais à esquerda desse nível.

O primeiro passo do NFDH consiste em colocar os itens em ordem decrescente de altura. Em seguida, empacotamos o primeiro item, na ordem estabelecida, no canto inferior esquerdo da faixa. Ao fazer isto, criamos o primeiro nível do empacotamento, que terá nesse momento $t_1 = a_1$ e $w_1 = l_1$. Assim, a altura desse primeiro nível será a_1.

Observe que, devido à ordenação dos itens, ao empacotar um item i, sabemos que sua altura é menor ou igual à altura de todos os níveis que foram criados anteriormente. Dessa forma, ao tentar empacotar um item i num nível já existente, somente precisamos verificar se a largura disponível neste nível é maior ou igual à l_i.

Cada item i seguinte é empacotado da seguinte maneira. Seja j o último nível criado. Se $l_i + w_j \leq L$, ou seja, se o item i cabe no nível j, empacotamos o item i na posição (w_j, t_{j-1}) da faixa[2] e fazemos $w_j = w_j + l_i$. Caso contrário, o item i não cabe no nível j. Precisamos então criar um novo nível imediatamente acima do nível j, fazendo $j = j + 1$, $w_j = l_i$ e $t_j = t_{j-1} + a_i$. Em seguida empacotamos o item i no canto inferior esquerdo deste novo nível, ou seja, na posição $(0, t_{j-1})$ da faixa. Descrevemos a seguir o algoritmo NFDH.

Coffman *et al.* [CGJT80] provaram que a razão de aproximação absoluta do NFDH é igual a 3. Estes autores demonstraram também que $\text{NFDH}(I) \leq 2\ \text{OPT}(I) + a_{max}$, onde a_{max} é a maior altura dentre todos os itens, e ainda que esta razão de aproximação

[2]Empacotar o item i na posição (w_j, t_{j-1}) da faixa significa posicionar o item i dentro da faixa de forma que o canto inferior esquerdo do item esteja na posição (w_j, t_{j-1}) da faixa.

Algoritmo 4.4 Next Fit Decreasing Height (NFDH)

Entrada: Uma instância $I = (L, l, a)$ do PEFB, onde $l = (l_1, \ldots, l_m)$ e $l = (a_1, \ldots, a_m)$.

Saída: Uma solução para I.

Ordene os itens em ordem decrescente de altura, obtendo $a_1 \geq \ldots \geq a_m$.
Faça $j = 0$, $t_0 =$, $w_0 = L$.
Para $i = 1$ até m
  Se $l_i + w_j > L$ `/* É preciso criar um novo nível */`
    Faça $j = j + 1$, $w_j = 0$ e $t_j = t_{j-1} + a_i$.
  Empacote o item i na posição $(w_j,\ t_{j-1})$ e faça $w_j = w_j + l_i$.
Devolva o empacotamento.

assintótica é justa.

Parece surpreendente que a razão de aproximação assintótica do NFDH seja igual à razão de aproximação absoluta do NF. No entanto, fazendo uma análise mais profunda, percebemos que o NFDH executa um procedimento bastante similar ao NF. Podemos considerar os níveis criados no NFDH como sendo os *bins* utilizados no NF. Dessa forma, o NFDH empacota os itens nos *bins* (níveis) da mesma forma que o NF. A diferença é que os níveis produzidos pelo NFDH têm altura variada e nos *bins* a altura não é relevante.

4.3.2 O Algoritmo *First Fit Decreasing Height*

Podemos obter um algoritmo para o PEFB com uma razão de aproximação assintótica melhor que a do NFDH introduzindo uma modificação semelhante àquela que foi feita no NF para se chegar ao FF. Suponha que tenham sido criados j níveis. Ao empacotar o item i, procuramos dentre os níveis existentes aquele de menor índice no qual cabe o item i, ou seja, procuramos o menor k tal que $k \leq j$ e $w_k + l_i \leq L$. Se existir tal k, empacotamos o item i na posição $(w_k,\ t_{k-1})$ da faixa e fazemos $w_k = w_k + l_i$. Caso contrário, criamos um novo nível, imediatamente acima do nível j: fazemos $j = j + 1$, $w_j = l_i$, $t_j = t_{j-1} + a_i$, e empacotamos o item i no canto inferior esquerdo deste novo nível. Este algoritmo é chamado de *First Fit Decreasing Height* (FFDH).

A diferença básica entre os algoritmos NFDH e FFDH é que no primeiro tentamos empacotar cada item somente no último nível que foi criado. Se isto não for possível, criamos um novo nível, ignorando possíveis espaços restantes nos níveis criados anteriormente. Já no FFDH, cada item é empacotado no nível de menor índice no qual ele caiba. Com isto, espaços remanescentes em todos os níveis podem vir a ser aproveitados. A descrição do

FFDH é dada a seguir.

Algoritmo 4.5 First Fit Decreasing Height (FFDH)

Entrada: Uma instância $I = (L, l, a)$ do PEFB, onde $l = (l_1, \ldots, l_m)$ e $l = (a_1, \ldots, a_m)$.

Saída: Uma solução para I.

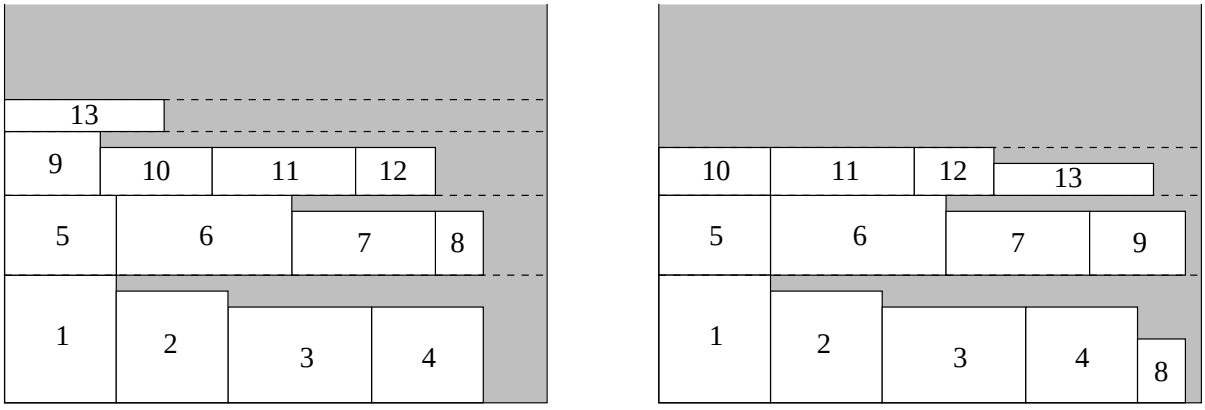
Ordene os itens em ordem decrescente de altura, obtendo $a_1 \geq \ldots \geq a_m$.
Faça $j = 0$, $t_0 =$, $w_0 = L$.
Para $i = 1$ até m
  Seja $k = min(\{h \mid 0 \leq h \leq j,\ l_i + w_h \leq L\} \cup \{j+1\})$.
  Se $k = j + 1$ faça $j = j + 1$, $w_j = 0$ e $t_j = t_{j-1} + a_i$. `/* Criando um novo nível */`
  Empacote o item i na posição $(w_j,\ t_{j-1})$ e faça $w_j = w_j + l_i$.
Devolva o empacotamento.

Assim como o FF e o FFD, os algoritmos NFDH e FFDH têm complexidade de tempo $\mathcal{O}(m \log m)$. Analogamente ao que ocorre para o NFDH e o FF, a razão de aproximação assintótica do FFDH é igual à do FF. Coffman *et al.* [CGJT80] provaram que $\mathrm{FFDH}(I) \leq \frac{17}{10}\ \mathrm{OPT}(I) + a_{max}$, para toda instância I do PEFB, sendo que esta razão de aproximação assintótica é justa. Sabe-se ainda que a razão de aproximação absoluta do FFDH é igual a 2,7. Estes autores mostraram também que, se todos os itens da instâncias I são quadrados, $\mathrm{FFDH}(I) \leq \frac{3}{2}\ \mathrm{OPT}(I) + 1$ e que esta razão de aproximação assintótica também é justa. Ademais, se os itens têm largura pequena em relação à largura da faixa, mais precisamente, se $l_i \leq \frac{L}{k}$ $(i = 1, \ldots, m)$, então a razão de aproximação assintótica do FFDH é $\frac{k+1}{k}$.

A Figura 4.2 mostra as soluções encontradas pelo NFDH e pelo FFDH para uma mesma instância. Os níveis são as regiões delimitadas por duas linhas pontilhadas consecutivas. A numeração indica a ordem em que os itens foram empacotados.

Existem ainda outros algoritmos para o PEFB, como por exemplo os algoritmos *Split Fit* (SF), *Mixed Algorithm* (MA) e *Up-Down* (UD). Estes três algoritmos se diferenciam do NFDH e do FFDH por produzir empacotamentos que não necessariamente estão estruturados em níveis. Em particular, o algoritmo UD pode produzir empacotamentos não guilhotináveis.

O algoritmo SF foi proposto por Coffman *et al.* [CGJT80] e sabe-se que $\mathrm{SF}(I) \leq \frac{3}{2}\ \mathrm{OPT}(I) + 2a_{max}$. Além disso, se todos os itens têm largura menor que $\frac{L}{k}$, então $\mathrm{SF}(I) \leq \frac{k+2}{k+1}\ \mathrm{OPT}(I) + 2$. O algoritmo MA foi proposto por Golan [Gol81], que provou a seguinte desigualdade: $\mathrm{MA}(I) \leq \frac{4}{3}\ \mathrm{OPT}(I) + \frac{127}{18}a_{max}$. O algoritmo UD foi proposto por Baker, Brown e Katseff [BBK81], os quais provaram que $\mathrm{UD}(I) \leq \frac{5}{4}\ \mathrm{OPT}(I) + \frac{53}{8}a_{max}$.



(a) Solução do NFDH (b) Solução do FFDH

Figura 4.2: Exemplos de solução do NFDH e do FFDH

Estes autores mostraram ainda que a razão de aproximação assintótica de seus respectivos algoritmos, aqui apresentadas, são justas.

Os algoritmos que citamos para o PEFB (NFDH, FFDH, SF, MA e UD) são todos *off-line*. Csirik e Woeginger [CW97] desenvolveram um algoritmo *on-line* para o PEFB, chamado de $\text{SHELF}_{H_M,p}$, que empacota os itens em níveis e cuja razão de aproximação assintótica pode ser tornar tão próxima de 1,691 quanto se queira, dependendo dos valores de M e p. Estes autores demonstraram também que qualquer algoritmo *on-line* para o PEFB, baseado em níveis, tem que ter razão de aproximação assintótica maior ou igual a 1,691.

Brown, Baker e Katseff [BBK82] mostraram que qualquer algoritmo *on-line* para o PEFB tem que ter razão de aproximação absoluta maior ou igual a 2. São conhecidos algoritmos de aproximação para o PEFB com razão de aproximação absoluta igual a 2 [Sch94, Ste97], mas estes algoritmos são do tipo *off-line*.

Recentemente, Kenyon e Rémila [KR00] apresentaram um FPAAS para o PEFB. Supondo que nenhum item tem altura maior que 1, eles provaram que para cada $\epsilon > 0$ existe um algoritmo KR_ϵ que é polinomial em $\frac{1}{\epsilon}$ e no tamanho da instância I tal que $\text{KR}_\epsilon(I) \leq (1+\epsilon)\ \text{OPT}(I) + \mathcal{O}(\frac{1}{\epsilon^2})$. Por exemplo, se $\epsilon = \frac{1}{4}$, KR_ϵ tem razão de aproximação assintótica igual à do algoritmo UD, no entanto a constante aditiva é 16. Além disso, KR_ϵ requer tempo $\mathcal{O}(n(\log n)\epsilon^{-8}\text{polylog}(\epsilon) + (\log^2 n)\epsilon^{-16}\text{polylog}(\epsilon)$. Esta complexidade de tempo e a constante aditiva associada à razão de aproximação deixam claro que este FPAAS não é mais vantajoso, em aplicações práticas, que os algoritmos para o PEFB vistos nesta seção.

Existem também algoritmos de aproximação para a variante do PEFB na qual os itens podem sofrer rotações. Chamaremos tal variante de PEFB^r. Miyazawa [Miy97] propôs os algoritmos *On-line Strip Packing using Rotations* (OSPR) e *Strip Packing using Rotations* (SPR) para o PEFB^r. O primeiro é um algoritmo *on-line* cuja razão de aproximação assintótica pode ser tão próxima de 1,75 quanto se queira. Já o algoritmo *off-line* SPR possui razão de aproximação assintótica aproximadamente igual a 1,6123.

Miyazawa [Miy97] desenvolveu também outros dois algoritmos de aproximação para a variante do PEFB^r onde todos os itens têm largura menor ou igual a $\frac{L}{k}$ (k fixo). Um destes algoritmos é *on-line* e sua razão de aproximação assintótica pode ser tão próxima de $\frac{k+1}{k}$ quanto se queira. O outro algoritmo é *on-line* e sua razão de aproximação assintótica é igual a $\frac{k+1}{k}$.

4.4 O Algoritmo *Hybrid First Fit*

Passamos a abordar o algoritmo *Hybrid First Fit* (HFF), proposto por Chung, Garey e Johnson [CGJ82]. O HFF encontra soluções aproximadas para o problema de corte de estoque bidimensional binário (PCEB$_2$). Vejamos novamente a definição do PCEB$_2$: dada uma quantidade ilimitada de placas retangulares de largura L e altura A, e uma lista de m itens retangulares, cada item i com largura $l_i \leq L$, altura $a_i \leq A$ e demanda igual a 1, queremos determinar como produzir uma unidade de cada item i utilizando o menor número possível de placas. Além disso, não é permitido que os itens sofram rotações.

O HFF foi proposto para o problema de empacotamento bidimensional binário[3]. Conforme discutimos na Seção 3.3, cortar uma placa de maneira a produzir itens de uma lista equivale a empacotar itens da lista dentro da placa. Portanto, em conformidade com a terminologia mais comumente adotada, descreveremos o HFF como sendo um algoritmo para empacotar os itens dentro das placas.

Representaremos um padrão de empacotamento (ou corte) bidimensional utilizando as mesmas convenções adotadas para representar um empacotamento para o PEFB (veja Seção 4.3), acrescentando apenas a restrição de que os itens devem ser empacotados na placa de maneira que não ultrapassem seu lado superior. Formalmente, um padrão de empacotamento bidimensional é uma coleção de itens $\wp$ que satisfaz as seguintes propriedades:

(a) Para todo item i em $\wp$, temos que $0 \leq x_i \leq L - l_i$ e $0 \leq y_i \leq A - a_i$ (o canto

[3]Este problema é análogo ao PCEB$_2$.

inferior esquerdo do item i está na posição $(x_i,\ y_i)$ da placa);

(b) Para todo par de itens distintos i, j em $\wp$, temos que $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$.

O HFF utiliza a seguinte estratégia: primeiramente, os itens são empacotados numa faixa de largura L utilizando-se o algoritmo FFDH. Suponha que o FFDH tenha produzido k níveis sendo h_j a altura do nível j $(j = 1, \ldots, k)$. Utilizamos em seguida o algoritmo FFD para resolver a instância do PEBB onde os *bins* têm comprimento A e cada item j tem comprimento h_j $(j = 1, \ldots, k)$. Isto equivale a empacotar os níveis produzidos pelo FFDH nas placas utilizando o algoritmo FFD.

Algoritmo 4.6 Hybrid First Fit (HFF)

Entrada: Uma instância $I = (L, A, l, a)$ do PCEB_2, onde $l = (l_1, \ldots, l_m)$ e $a = (a_1, \ldots, a_m)$.

Saída: Uma solução para I.

Execute o FFDH com parâmetros $L, l = (l_1, \ldots, l_m), a = (a_1, \ldots, a_m)$, obtendo um empacotamento com k níveis, onde cada nível j tem altura h_j $(j = 1, \ldots, k)$.

Execute o FFD com parâmetros $A, l = (h_1, \ldots, h_k)$, obtendo um empacotamento que utiliza j *bins*.

Empacote os k níveis produzidos pelo FFDH em j placas, conforme a solução produzida pelo FFD.

Devolva o empacotamento.

É fácil perceber que o HFF tem complexidade de tempo $\mathcal{O}(m \log m)$. Chung, Garey e Johnson [CGJ82], provaram que $\text{HFF}(I) \leq \frac{17}{8}\ \text{OPT}(I) + 5$. No entanto, eles não conseguiram provar que a razão de aproximação assintótica $\frac{17}{8}$ deste algoritmo é justa. Sabe-se porém que $2,022 \approx \frac{91}{45} \leq R^{\infty}_{HFF} \leq \frac{17}{8} = 2,125$. Recentemente, Caprara [Cap02] mostrou que a razão de aproximação assintótica do HFF é menor ou igual a $\frac{11}{9} \times \frac{17}{10} \approx 2,077$. Isto responde a questão levantada por Chung, Garey e Johnson sobre a possibilidade de se provar que a razão de aproximação assintótica do HFF é igual à multiplicação das razões de aproximação assintótica dos algoritmos FFD e FFDH, que o HFF utiliza como sub-rotina e que possuem razão de aproximação assintótica igual a $\frac{11}{9}$ e $\frac{17}{10}$, respectivamente.

Observe ainda que o HFF produz apenas padrões guilhotináveis [4]. Dessa forma, o HFF também pode ser utilizado para resolver o problema de corte de guilhotina bidimensional

[4] Um padrão produzido pelo HFF pode ser guilhotinado em 3 estágios: o primeiro estágio (horizontal) serve para separar os níveis, o segundo para separar os itens dentro de cada nível e o terceiro estágio para separar os itens de regiões desperdiçadas acima deles.

binário ($PCGB_2$). Além disso, por usar o FFDH e o FFD como sub-rotina, o HFF é um algoritmo *off-line*.

Existem ainda outros algoritmos de aproximação para o $PCEB_2$. Um deles é o algoritmo *Hybrid Next Fit* (HNF). Dada uma instância do $PCEB_2$, denotemos por l_{max} e a_{max} a maior largura e a maior altura dentre todos os itens, respectivamente. Frenk e Galambos [FG87] demonstraram ser válida uma desigualdade[5] que fornece a razão de aproximação assintótica do HNF em função de l_{max} e a_{max}. Por exemplo, se $\frac{1}{5} < l_{max} \leq \frac{1}{4}$ e $\frac{1}{5} < a_{max} \leq \frac{1}{4}$, a razão de aproximação assintótica do HNF é aproximadamente 1,644. No caso geral, a razão de aproximação assintótica do HNF é aproximadamente 3,382, sendo que esta razão é justa.

Tanto o HFF quanto o HNF são algoritmos *off-line*. Foram propostos diversos algoritmos *on-line* para o $PCEB_2$. Podemos citar o algoritmo proposto por Coppersmith e Raghavan [CR89] que possui razão de aproximação assintótica não maior que 3,25, e os algoritmos desenvolvidos por Li e Cheng [LC90a] e Csirik e van Vliet [CV93], que possuem razão de aproximação assintótica igual a 2,86. Sabe-se ainda que qualquer algoritmo *on-line* para o $PCEB_2$ tem que ter razão de aproximação assintótica maior ou igual a 1,907 [BvW96].

Caprara [Cap02] observou que o FPAAS proposto por Kenyon e Rémila [KR00] para o PEFB, fornece algoritmos para o $PCEB_2$ com razão de aproximação assintótica $2 + \epsilon$ ($\epsilon > 0$). No entanto, os padrões produzidos podem não ser guilhotináveis. Caprara, Lodi e Monacci [CLM02] desenvolveram um PAAS para a variante do $PCEB_2$ onde os padrões devem ser guilhotináveis em dois estágios.

Existem também algoritmos de aproximação para a variante do $PCEB_2$ na qual os itens podem sofrer rotações. Tal variante será chamada de $PCEB_2^r$. Miyazawa [Miy97] propôs os algoritmos *On-line Bidimensional Packing* (OBI) e *Bidimensional Packing* ($BI_{k,\epsilon}$) para o $PCEB_2^r$. O algoritmo OBI é uma adaptação do algoritmo proposto por Coppersmith e Raghavan [CR89] para o $PCEB_2$ e possui razão de aproximação assintótica igual a 3,25. A razão de aproximação assintótica do algoritmo (*off-line*) $BI_{k,\epsilon}$ pode ser tão próxima de 2,639 quanto se queira. O algoritmo $BI_{k,\epsilon}$ utiliza o PAAS de Fernandez de la Vega e Lueker [FL81] como sub-rotina e sua implementação não é trivial.

Uma outra variante do $PCEB_2$ que tem sido estudada é o problema de empacotamento de quadrados em quadrados binário (PEQB). Recentemente, Kohayakawa, Miyazawa, Raghavan e Wakabayashi [KMRW01] apresentaram um algoritmo de aproximação

[5]Para entender tal desigualdade precisaríamos introduzir várias definições. Por brevidade, decidimos não transcrevê-la.

para problema de empacotamento de cubos N-dimensional. Estes autores mostraram que tal algoritmo têm razão de aproximação assintótica tão próxima de $2 - \left(\frac{2}{3}\right)^N$ quanto se queira. Para o caso bidimensional, a razão de aproximação assintótica deste algoritmo é de aproximadamente 1,555, sendo esta a melhor razão conhecida até o momento.

Diversos pesquisadores têm feito análises probabilísticas de vários dos algoritmos que citamos neste capítulo [OMW84, CGF$^+$86, CLR88, CJSW97, FvR97, CCG$^+$02]. Para alguns deles foram determinadas razões de aproximação esperadas. Detalhes a este respeito podem ser encontradas nas referências fornecidas.

Na Tabela 4.1 apresentamos uma lista contendo a maior parte dos algoritmos e esquemas de aproximação que citamos neste capítulo, indicando o problema para o qual foram desenvolvidos, a razão de aproximação (α), a constante aditiva (β) e as principais referências bibliográficas relacionadas com cada algoritmo.

Algoritmo	Problema	Tipo	α	β	Referência
NF	PEBB	*on-line*	2	—	[Joh73]
FF	PEBB	*on-line*	1,7	1	[Joh73, GGJY76]
BF	PEBB	*on-line*	1,7	2	[JDU$^+$74]
NFD	PEBB	*off-line*	1,691...	3	[BC81]
FFD	PEBB	*off-line*	1,222...	1	[Joh73, Yue91]
BFD	PEBB	*off-line*	1,222...	4	[JDU$^+$74]
MFFD	PEBB	*off-line*	1,1833...	$\frac{31}{6}$	[JG85]
FL_ϵ	PEBB	*off-line*	$1+\epsilon$	$\frac{1}{\epsilon}$	[FL81, CGJ97]
KK	PEBB	*off-line*	1	$\mathcal{O}(\log^2 \mathrm{OPT}(I))$	[KK82]
NFDH	PEFB	*off-line*	2	a_{max}	[CGJT80]
FFDH	PEFB	*off-line*	1,7	a_{max}	[CGJT80]
$\mathrm{SHELF}_{H_M,p}$	PEFB	*on-line*	$\asymp$ 1,691	$\frac{M}{1-p}$	[CW97]
SF	PEFB	*off-line*	1,5	$2a_{max}$	[CGJT80]
MA	PEFB	*off-line*	1,333...	$\frac{127}{18}a_{max}$	[Gol81]
UD	PEFB	*off-line*	1,25	$\frac{53}{8}a_{max}$	[BBK81]
KR_ϵ	PEFB	*off-line*	$1+\epsilon$	$\mathcal{O}(\frac{1}{\epsilon^2})$	[KR00]
OSPR	PEFB^r	*on-line*	$\asymp 1,75\ldots$	$\frac{2a_{max}}{1-p}$	[Miy97]
SPR	PEFB^r	*off-line*	1,6123...	$4a_{max}$	[Miy97]
HNF	PCEB_2	*off-line*	3,382	9	[FG87]
HFF	PCEB_2	*off-line*	2,077...	?	[CGJ82, Cap02]
OBI	PCEB_2^r	*on-line*	3,25	10	[Miy97]
$\mathrm{BI}_{k,\epsilon}$	PCEB_2^r	*off-line*	$\asymp 2,639\ldots$	$\mathcal{O}(k+\frac{1}{\epsilon})$	[Miy97]

Tabela 4.1: Alguns algoritmos e esquemas de aproximação para problemas de empacotamento bidimensional.

Capítulo 5

Novos Algoritmos de Aproximação

5.1 Introdução

Neste capítulo, propomos algoritmos de aproximação para o problema de corte de estoque bidimensional com demandas (PCED_2). Relembremos a definição deste problema: dada uma quantidade ilimitada de placas retangulares de largura L e altura A, e uma lista de m itens retangulares, cada item i com largura $l_i \leq L$, altura $a_i \leq A$ e demanda d_i, queremos determinar como empacotar d_i cópias de cada item i da lista utilizando o menor número possível de placas.

Apresentaremos também um algoritmo de aproximação para a variante do PCED_2 onde todas as placas e itens são quadrados, que é chamada de problema de empacotamento de quadrados em quadrados com demandas (PEQD). Até onde sabemos, esses são os primeiros algoritmos de aproximação propostos para estes problemas.

Finalmente, propomos um algoritmo de aproximação bastante simples para a variante do PCED_2 na qual rotações são permitidas e todos os itens possuem demanda igual a 1, e que é chamada de problema de corte de estoque bidimensional binário com rotações (PCEB_2^r).

Na Seção 4.4 apresentamos diversos algoritmos de aproximação para o PCEB_2, tais como o HFF e o HNF. Podemos transformar uma instância do PCED_2 numa instância do PCEB_2 simplesmente substituindo cada item i com demanda d_i por d_i cópias do item i. Por exemplo, a instância ($L = 10$, $A = 8$, $l = \{3, 2, 4\}$, $a = \{2, 5, 1\}$, $d = \{5, 1, 8\}$) do PCED_2 seria transformada na seguinte instância do PCEB_2: ($L = 10$, $A = 8$, $l = \{3, 3, 3, 3, 3, 2, 4, 4, 4, 4, 4, 4, 4, 4\}$, $a = \{2, 2, 2, 2, 2, 5, 1, 1, 1, 1, 1, 1, 1, 1\}$).

No entanto, essa transformação pode levar uma instância I do PCED_2 de tamanho

m a uma instância I' do PCEB_2 de tamanho exponencial em m. Dessa forma, o HFF levaria tempo exponencial em m para resolver I'. Consequentemente um algoritmo para o PCGD_2 que realiza a transformação indicada no parágrafo anterior e aplica o HFF não é um algoritmo de aproximação para o PCED_2. Na verdade, todo algoritmo para o PCED_2 que realize pelo menos uma operação para cada retângulo empacotado necessariamente vai requerer tempo exponencial em relação ao tamanho da instância, no pior caso.

Assim, ao projetar um algoritmo para o PCED_2, temos que ter cuidado também com a representação da resposta gerada pelo algoritmo. Dada uma instância do PCED_2 com m itens, podemos precisar de uma quantidade de placas exponencial em m. Ainda que precisemos gastar uma quantidade limitada de memória para representar cada placa usada no empacotamento, podemos precisar de uma quantidade exponencial de memória para representar a solução. Consequentemente, o algoritmo levaria tempo exponencial apenas para escrever a resposta.

Dessa forma, para descrever eficientemente uma solução para uma instância do PCED_2, precisamos nos limitar a uma quantidade polinomial de padrões e especificar a quantidade de vezes que cada padrão deve ser utilizado. Mas isto ainda não é suficiente. Observe que, se as dimensões dos itens forem muito pequenas em relação às dimensões das placas, num padrão poderemos ter uma quantidade exponencial de retângulos. Se, ao descrever um padrão, precisarmos gastar uma quantidade qualquer de memória para representar cada item contido no padrão, poderemos precisar de uma quantidade exponencial de memória para representar o padrão, e então o algoritmo precisaria de tempo exponencial.

Portanto, se desejamos obter um algoritmo polinomial para o PCED_2 temos que:

(a) utilizar uma quantidade polinomial de padrões (e especificar a quantidade de vezes que cada padrão deve ser utilizado).

(b) representar cada padrão com uma quantidade polinomial de memória.

Para atender estes dois requisitos faremos uso de padrões homogêneos. É deste assunto que trata a próxima seção.

5.2 Utilizando Padrões Homogêneos

Dizemos que um padrão é *homogêneo* se ele contém apenas itens de mesmas dimensões. Denotamos por $H(L, A, l, a, d)$ o padrão homogêneo contendo d cópias de um item de largura l e altura a numa placa de largura L e altura A. Claramente, para que $H(L, A, l, a, d)$

seja factível é preciso que $l \leq L$, $a \leq A$ e $d \leq \lfloor \frac{L}{l} \rfloor \lfloor \frac{A}{a} \rfloor$. Padrões que obedecem tais restrições serão chamados de *viáveis*. Em geral, estaremos sempre nos referindo a padrões viáveis.

Empacotamos os itens em $H(L, A, l, a, d)$ da seguinte maneira. Fazemos $x = \lfloor \frac{L}{l} \rfloor$, para j variando de 0 até $d-1$, empacotamos a j-ésima cópia do item na posição $(l(j \bmod x), a\lfloor \frac{j}{x} \rfloor)$. Observe que o empacotamento terá $\lceil \frac{d}{x} \rceil$ níveis, cada nível contendo x itens (exceto possivelmente o último). Um exemplo do resultado deste procedimento é ilustrado pela Figura 5.1 na qual é exibido o $H(34, 21, 8, 6, 10)$. Note que este padrão é guilhotinável.

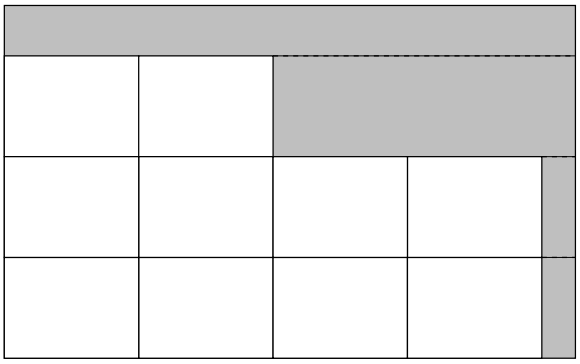


Figura 5.1: $H(34, 21, 8, 6, 10)$

Podemos representar padrões da forma $H(L, A, l, a, d)$ com uma quantidade de memória polinomial, visto que as posições dos retângulos no padrão não precisam ser armazenadas já que podem ser calculadas de acordo com o procedimento que acabamos de descrever. Ademais, tais padrões são guilhotináveis. Sendo assim, o PCED_2H também resolve o problema de corte de guilhotina bidimensional com demandas (PCGD_2).

A seguir descrevemos o algoritmo de aproximação PCED_2, que é baseado na utilização de padrões homogêneos e que resolve o PCED_2.

Podemos relacionar o valor da solução encontrada pelo algoritmo PCED_2H com o valor de uma solução ótima. Antes, vejamos algumas definições. Seja $\mathcal{P}$ um padrão de empacotamento bidimensional. Denotamos por $\mathcal{A}(\mathcal{P})$ a soma das áreas dos itens contidos no padrão $\mathcal{P}$, ou seja, a área aproveitada no padrão $\mathcal{P}$. Denotamos por $\mathcal{D}(\mathcal{P})$ a área desperdiçada no padrão $\mathcal{P}$.

Chamamos de padrões homogêneos *maximais* os padrões homogêneos que contêm a maior quantidade possível de cópias do item. Observe que $H(L, A, l, a, \lfloor \frac{L}{l} \rfloor \lfloor \frac{A}{a} \rfloor))$ é um padrão maximal.

Algoritmo 5.1 PCED_2H

Entrada: Uma instância $I = (L, A, l, a, d)$ do PCED_2, onde $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I.

1 Para $i = 1$ até m

1.1 $x_i = \lfloor \frac{L}{l_i} \rfloor$, $y_i = \lfloor \frac{A}{a_i} \rfloor$ e $z_i = \lfloor \frac{d_i}{x_i y_i} \rfloor$.

1.2 Se $z_i > 0$ então empacote $x_i y_i z_i$ cópias do item i utilizando z_i vezes o padrão $H(L, A, l_i, a_i, x_i y_i)$.

1.3 Se $d_i > x_i y_i z_i$ então empacote $d_i - x_i y_i z_i$ cópias do item i utilizando uma vez o padrão $H(L, A, l_i, a_i, d_i - x_i y_i z_i)$.

2 Devolva o empacotamento.

Lema 5.2.1 *Seja $\mathcal{P}$ o padrão viável $H(L, A, l, a, \lfloor \frac{L}{l} \rfloor \lfloor \frac{A}{a} \rfloor)$. Então $\mathcal{A}(\mathcal{P}) > \frac{LA}{4}$.*

Prova. Note que $\lfloor \frac{L}{l} \rfloor l > (\frac{L}{l} - 1)l > L - l$. Ademais, $\lfloor \frac{L}{l} \rfloor l \geq l$. Dessa forma, $2\lfloor \frac{L}{l} \rfloor l > L - l + l = L$, logo $\lfloor \frac{L}{l} \rfloor l > \frac{L}{2}$. Analogamente, $\lfloor \frac{A}{a} \rfloor a > \frac{A}{2}$. Disto decorre que:

$$\mathcal{A}(\mathcal{P}) = la \left\lfloor \frac{L}{l} \right\rfloor \left\lfloor \frac{A}{a} \right\rfloor = \left\lfloor \frac{L}{l} \right\rfloor l \left\lfloor \frac{A}{a} \right\rfloor a > \frac{L}{2} \frac{A}{2} = \frac{LA}{4}.$$

■

Dessa forma, os padrões (maximais) utilizados no passo 1.2 do algoritmo PCED_2H aproveitam pelo menos $\frac{1}{4}$ da área da placa. Já a área aproveitada nos padrões utilizados no passo 1.3 (não-maximais), ainda que não-nula, não nos fornece uma garantia de ocupação relativamente à área da placa. Baseados nestas observações, podemos concluir o seguinte resultado.

Teorema 5.2.2 $\text{PCED}_2\text{H}(I) \leq 4\ \text{OPT}(I) + m$, *para toda instância I do $PCED_2$.*

Prova. Observe que a solução encontrada pelo PCED_2H utiliza no máximo m padrões não-maximais. Disto decorre que:

$$\text{PCED}_2\text{H}(I) \leq \sum_{i=1}^{m} z_i + m = \sum_{i=1}^{m} \frac{LAz_i}{LA} + m < \sum_{i=1}^{m} \frac{4\mathcal{A}\big(H(L, A, l_i, a_i, x_i y_i)\big) z_i}{LA} + m \leq$$

$$4 \sum_{i=1}^{m} \frac{l_i a_i d_i}{LA} + m \leq 4\ \text{OPT}(I) + m.$$

■

Pelo Teorema 5.2.2 percebemos que o valor da solução produzida pelo PCED_2H pode ser bem distante do valor de uma solução ótima. Considere a seguinte família de instâncias: m itens, todos eles com largura e altura (distintas) menores que 1 e demanda igual a 1, e placas de largura e altura m. Claramente é possível empacotar todos os itens utilizando uma única placa. No entanto, o algoritmo PCED_2H produz uma solução de valor m. Para uma instância I desta família, $\lim_{m\to\infty} \frac{\text{PCED}_2\text{H}(I)}{\text{OPT}(I)} = \infty$.

Isto significa que o algoritmo PCED_2H pode produzir uma solução de valor infinitamente maior que o valor de uma solução ótima. Na seção seguinte introduzimos os conceitos de bloco homogêneo e de padrão semi-homogêneo e mostramos como utilizá-los para produzir um algoritmo de aproximação para o PCED_2 com razão de aproximação absoluta igual a 4.

5.3 Utilizando Padrões Semi-homogêneos

Dado um padrão de empacotamento bidimensional, chamamos de *blocos homogêneos* um retângulo minimal que contém todos os itens de mesmas dimensões dentro do padrão. Dizemos que um padrão de empacotamento bidimensional é *semi-homogêneo* se os blocos homogêneos contidos no padrão não se sobrepõem. Na Figura 5.2 temos à esquerda um padrão semi-homogêneo (a) e à direita um padrão que não é semi-homogêneo (b).

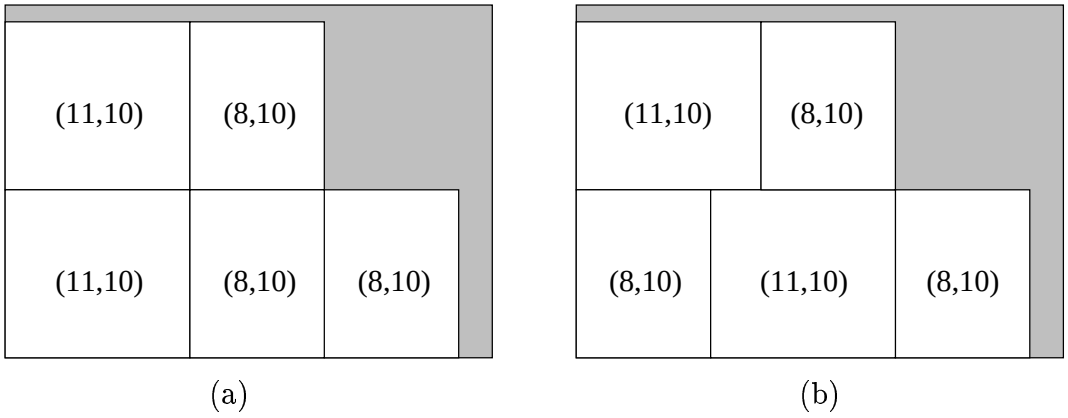


Figura 5.2: Um padrão semi-homogêneo e um padrão que não é semi-homogêneo

Observe que num padrão semi-homogêneo, cada bloco homogêneo é equivalente a um padrão homogêneo. Sendo assim, cada bloco homogêneo pode ser descrito com uma

quantidade de memória polinomial. Ademais, a quantidade de blocos homogêneos num padrão semi-homogêneo é limitada por m, a quantidade de itens. Portanto, um padrão semi-homogêneo também pode ser descrito com uma quantidade de memória polinomial.

Propomos a seguir um algoritmo de aproximação para o PCED$_2$ que utiliza padrões homogêneos e semi-homogêneos. A idéia básica deste algoritmo é primeiramente utilizar padrões homogêneos que aproveitam pelo menos $\frac{1}{4}$ da área da placa. A seguir, para cada item cuja demanda não tenha sido totalmente atendida com os padrões homogêneos, utilizamos no máximo 2 blocos homogêneos. Finalmente, estes blocos homogêneos são empacotados nas placas utilizando-se o algoritmo HFF, descrito na Seção 4.4. Note que teremos no máximo $2m$ blocos homogêneos. Isto garante que o HFF vai requerer tempo polinomial em m.

Algoritmo 5.2 PCED$_2$SH

Entrada: Uma instância $I = (L, A, l, a, d)$ do PCED$_2$, onde $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I.

1 $n = 0$.
2 Para $i = 1$ até m
  2.1 $x_i = \lfloor \frac{L}{l_i} \rfloor$, $y_i = \lfloor \frac{A}{a_i} \rfloor$ e $z_i = \lfloor \frac{d_i}{x_i y_i} \rfloor$.
  2.2 Se $z_i > 0$ então empacote $x_i y_i z_i$ cópias do item i utilizando z_i vezes o padrão $H(L, A, l_i, a_i, x_i y_i)$.
  2.3 $d'_i = d_i - x_i y_i z_i$.
  2.4 Se $l_i a_i d'_i \geq \frac{LA}{4}$ então empacote d'_i cópias do item i utilizando uma vez o padrão $H(L, A, l_i, a_i, d'_i)$.
  2.5 Se $d'_i > 0$ e $l_i a_i d'_i < \frac{LA}{4}$
    2.5.1 $y'_i = \lfloor \frac{d'_i}{x_i} \rfloor$, $n = n + 1$, $l'_n = l_i x_i$ e $a'_n = a_i y'_i$.
    2.5.2 Se $x_i y'_i < d'_i$ então $n = n + 1$, $l'_n = l_i(d'_i - x_i y'_i)$ e $a'_n = a_i$.
3 Se $n > 0$ então resolva a instância $I' = (L, A, l' = (l'_1, \ldots, l'_n), a' = (a'_1, \ldots, a'_n))$ com o algoritmo HFF.
4 Devolva o empacotamento.

Vamos agora mostrar que a razão de aproximação absoluta do PCED$_2$SH é 4.

Teorema 5.3.1 PCED$_2$SH$(I) \leq 4$ OPT(I), *para toda instância I do PCED$_2$.*

Prova. Seja PCED$_2$SH$(I) = k$. Vamos dividir a prova em dois casos, dependendo do valor de n ao final da execução do algoritmo PCED$_2$SH.

Caso 1: $n = 0$. Neste caso, o algoritmo HFF não é utilizado, e todos os padrões utilizados na solução são definidos nos passos 2.2 e 2.4 do PCED_2SH. Pelo Lema 5.2.1, os padrões utilizados no passo 2.2 do algoritmo PCED_2SH aproveitam pelo menos $\frac{1}{4}$ da área da placa. Claramente, isto também é válido para os padrões utilizados no passo 2.4 do PCED_2SH. Dessa forma, $\sum_{i=1}^{m} l_i a_i d_i \geq \frac{kLA}{4}$. Disto decorre que,

$$\text{PCED}_2\text{SH}(I) = k = \frac{4}{LA}\frac{kLA}{4} \leq 4\frac{\sum_{i=1}^{m} l_i a_i d_i}{LA} \leq 4\ \text{OPT}(I).$$

Caso 2: $n > 0$. Primeiramente observamos que $n \leq 2m$, e portanto o HFF aplicado à instância I' requer tempo polinomial em m.

Vamos considerar as placas utilizadas pelo HFF indexadas de 1 a r, conforme a ordem em que começaram a ser aproveitadas. Para facilitar a compreensão da prova, dividiremos este caso em 2 subcasos.

Subcaso 2.1: $r \leq 3$. Então, $\sum_{i=1}^{m} l_i a_i d_i > \frac{(k-r)LA}{4}$. Observe que,

$$\text{OPT}(I) \geq \left\lceil \frac{\sum_{i=1}^{m} l_i a_i d_i}{LA} \right\rceil > \left\lceil \frac{(k-r)LA}{4LA} \right\rceil = \left\lceil \frac{k-r}{4} \right\rceil.$$

Como $\text{OPT}(I)$ é inteiro, temos que:

$$\text{OPT}(I) \geq \left\lceil \frac{k-r}{4} \right\rceil + 1 \geq \frac{k-r}{4} + 1 \geq \frac{k-3}{4} + 1 = \frac{k+1}{4} > \frac{k}{4} = \frac{\text{PCED}_2\text{SH}(I)}{4}.$$

Subcaso 2.2: $r \geq 4$. Vamos considerar os níveis produzidos no HFF indexados de 1 a t, conforme a ordem em que foram criados. Seja h_i a altura do nível i, A_i a área aproveitada no nível i, b_i a largura do nível i, e c_i a largura do primeiro item de I' contido no nível i (o item mais à esquerda). Vamos mostrar que a seguinte desigualdade é válida.

$$\sum_{i=1}^{t} l_i a_i d'_i > \frac{(r-1)LA}{4}. \tag{5.1}$$

Note que, para $1 \leq i < j \leq t$, temos que $b_i + c_j > L$ e $h_i \geq h_j$. Consequentemente, para $1 \leq i < j \leq t$,

$$A_i + A_j \geq h_j b_i + h_j c_j = h_j(b_i + c_j) > h_j L.$$

Seja s_j a soma das alturas dos níveis contidos na placa j. Note que, para $1 \leq j < r$, temos que $s_j > \frac{A}{2}$. Portanto,

$$2\sum_{i=1}^{t} A_i = 2A_1 + \ldots + 2A_t > A_1 + h_2 L + \ldots + h_t L + A_t > (h_2 + \ldots + h_t)L + h_t L$$

$$\geq L\sum_{j=2}^{r} s_j + h_t L \geq \frac{LA}{2}(r-2) + s_r L + h_t L \geq \frac{LA}{2}(r-2) + 2h_t L.$$

Se $h_t \geq \frac{A}{4}$ então $2h_t L \geq \frac{LA}{2}$, e portanto, $2\sum_{i=1}^{t} A_i > \frac{LA}{2}(r-1)$, ou seja, a área aproveitada nas placas utilizadas pelo HFF é maior que $(r-1)\frac{LA}{4}$ e portanto (5.1) é válida.

Suponha agora que $h_t < \frac{A}{4}$. Portanto, para $1 \leq j < r$, temos que $s_j > \frac{3A}{4}$. Disto decorre que,

$$2\sum_{i=1}^{t} A_i = 2A_1 + \ldots + 2A_t > (h_2 + \ldots + h_t)L \geq L\sum_{j=2}^{r} s_j > \frac{3LA}{4}(r-2) + s_r L$$

$$> \frac{3LA}{4}(r-1) - \frac{3LA}{4} \geq \frac{LA}{2}(r-1) + \frac{LA}{4}(r-1) - \frac{3LA}{4} \geq \frac{LA}{2}(r-1).$$

Novamente, concluímos que (5.1) é válida. Sejam $\mathcal{J} = \{i \mid d'_i = 0$ ou $(d'_i > 0$ e $l_i a_i d'_i \geq \frac{LA}{4})\}$ e $\mathcal{R} = \{i \mid d'_i > 0$ e $l_i a_i d'_i < \frac{LA}{4})\}$. Cada um dos itens $i \in \mathcal{J}$ tem sua demanda d_i atendida pelas placas utilizadas nos passos 2.2 e/ou 2.3. Por outro lado, cada um dos itens $i \in \mathcal{R}$ tem sua demanda $d_i = x_i y_i z_i + d'_i$ atendida pelas z_i placas utilizadas no passo 2.2 (se $z_i > 0$) mais as placas adicionais utilizadas pelo algoritmo HFF.

Assim, as $k-r$ placas geradas nos passos 2.3 e 2.3 atendem completamente as demandas d_i dos itens $i \in \mathcal{J}$ e parte das demandas (precisamente $x_i y_i z_i$) dos itens $i \in \mathcal{R}$. Como essas placas têm pelo menos $\frac{1}{4}$ de sua área ocupada, segue que

$$\sum_{i \in \mathcal{J}} l_i a_i d_i + \sum_{i \in \mathcal{R}} l_i a_i x_i y_i z_i \geq \frac{(k-r)LA}{4}. \tag{5.2}$$

Como o algoritmo HFF utiliza r placas para atender as demandas restantes d'_i dos itens $i \in \mathcal{R}$, temos que $\sum_{i \in \mathcal{R}} l_i a_i d'_i = \sum_{i=1}^{t} A_i$, e portanto, pela desigualdade (5.1),

$$\sum_{i \in \mathcal{R}} l_i a_i d'_i > \frac{(r-1)LA}{4}. \tag{5.3}$$

Usando (5.2) e (5.3) temos que

$$\text{PCED}_2\text{SH}(I) - 1 = k - 1 = (k-r) + (r-1) = \frac{4}{LA}\left[\frac{(k-r)LA}{4} + 1 + \frac{(r-1)LA}{4}\right] <$$

$$\frac{4}{LA}\left[\sum_{i \in \mathcal{J}} l_i a_i d_i + \sum_{i \in \mathcal{R}} l_i a_i x_i y_i z_i + \sum_{i \in \mathcal{R}} l_i a_i d'_i\right] = \frac{4}{LA}\left[\sum_{i \in \mathcal{J}} l_i a_i d_i + \sum_{i \in \mathcal{R}} l_i a_i (x_i y_i z_i + d'_i)\right] =$$

$$\frac{4}{LA}\sum_{i=1}^{m} l_i a_i d_i + 1 \leq 4\frac{\sum_{i=1}^{m} l_i a_i d_i}{LA} + 1 \leq 4\ \text{OPT}(I) + 1.$$

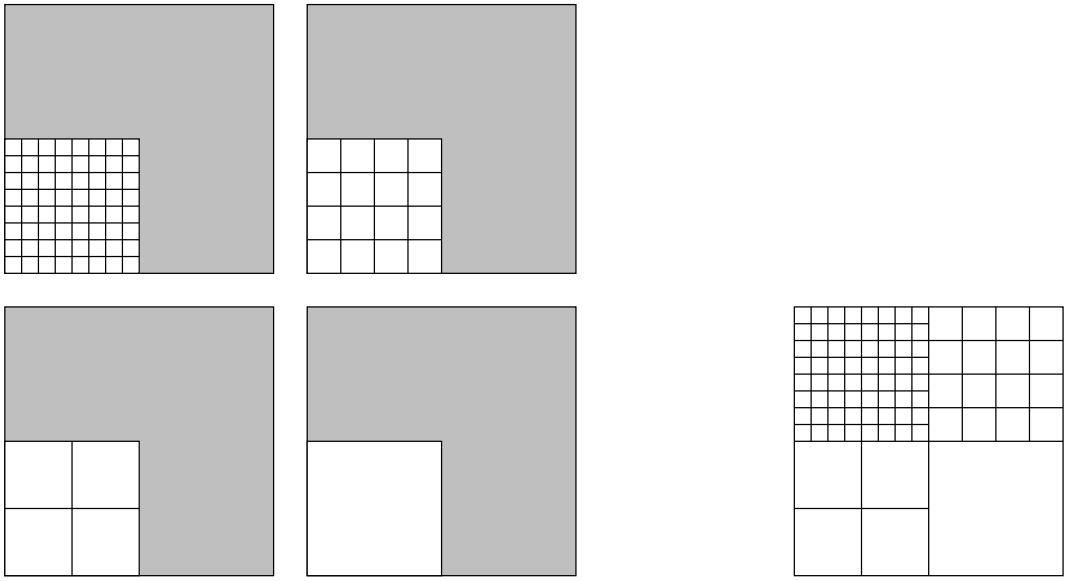
Como $\text{PCED}_2\text{SH}(I)$ e $\text{OPT}(I)$ são inteiros, temos que $\text{PCED}_2\text{SH}(I) \leq 4\ \text{OPT}(I)$.

■

É fácil perceber que a razão de aproximação absoluta estabelecida pelo Teorema 5.3.1 é justa. Considere a seguinte família de instâncias I_m do PCED_2, definida para m múltiplo de 4: $L = A = 2^m$; $l_i = a_i = 2^{i-1}$ e $d_i = 2^{2m-2i}$ $(i = 1, \ldots, m)$. Para cada item i, o algoritmo PCED_2SH vai utilizar, no passo 2.4, um padrão $H(2^m, 2^m, 2^{i-1}, 2^{i-1}, 2^{2m-2i})$. A solução produzida pelo PCED_2SH terá então valor m. Note que podemos empacotar as 2^{2m-2i} cópias de cada item i utilizando um quadrado de lado 2^{m-1}, sem que haja desperdício de área. Sendo assim, a solução ótima para as instâncias dessa família tem valor $\frac{m}{4}$. Na Figura 5.3 indicamos em (a) a solução produzida pelo algoritmo PCED_2SH, e em (b) uma solução ótima para a instância I_4.

5.4 Empacotando Quadrados em Quadrados

É possível utilizar o algoritmo PCED_2SH para resolver uma variante do PCED_2 que chamaremos de problema de empacotamento de quadrados em quadrados com demandas (PEQD). Numa instância do PEQD os itens e as placas são quadrados. Cada item i tem lado l_i e demanda d_i e as placas têm lado L. Observe que o PEQD é um caso particular



(a) Solução do $PCED_2SH$ (b) Uma solução ótima

Figura 5.3: Exemplo de solução do $PCED_2SH$

do $PCED_2$. A família de instâncias que exibimos no final da Seção 5.3 é constituída somente de quadrados, portanto a razão de aproximação absoluta do $PCED_2SH$ para resolver instâncias do PEQD também é justa.

Podemos introduzir algumas modificações no algoritmo $PCED_2SH$ de modo a obter um algoritmo para o PEQD com razão de aproximação melhor que 4. Desta vez, objetivamos que a área ocupada em cada placa (com exceção de algumas delas) seja pelo menos $\frac{4L^2}{9}$. Neste caso, as alterações consistem em trocar a condição do passo 2.4 para $l_i^2 d'_i \geq \frac{4L^2}{9}$. No passo 2.5, quando $l_i^2 d'_i < \frac{4L^2}{9}$, definimos os itens da instância I' de forma que suas larguras e alturas sejam menores ou iguais a $\frac{L}{2}$. Por exemplo, dada uma instância I com $L = 59$, a Figura 5.4 mostra os itens da instância I' correspondente que seriam criados por um item de lado $l_1 = 10$ e demanda $d'_1 = 15$. Observe que todos esses 6 itens de I' teriam largura e altura menor que $\frac{L}{2}$. A seguir descrevemos a adaptação do algoritmo $PCED_2SH$ para o PEQD.

Observe que, para um valor fixo de i, as condições dos passos 2.5.2 e 2.5.3 não podem ser ambas satisfeitas. Portanto, para cada item de I teremos no máximo 9 itens em I' [1]. Isto garante que o HFF pode resolver I' em tempo polinomial em m.

[1] Na verdade, uma análise cuidadosa do algoritmo nos permite concluir que para cada item de I teremos no máximo 7 itens em I'.

Algoritmo 5.3 PEQDSH

Entrada: Uma instância $I = (L, l, d)$ do PEQD, onde $l = (l_1, \ldots, l_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I.

1 $n = 1$.

2 Para $i = 1$ até m

2.1 $x_i = \lfloor \frac{L}{l_i} \rfloor$ e $z_i = \lfloor \frac{d_i}{x_i^2} \rfloor$.

2.2 Se $z_i > 0$ então empacote $x_i^2 z_i$ cópias do item i utilizando z_i vezes o padrão $H(L, L, l_i, l_i, x_i^2)$.

2.3 $d'_i = d_i - x_i^2 z_i$.

2.4 Se $l_i^2 d'_i \geq \frac{4L^2}{9}$ empacote d'_i cópias do item i utilizando o padrão $H(L, A, l_i, a_i, d'_i)$.

2.5 Se $d'_i > 0$ e $l_i^2 d'_i < \frac{4L^2}{9}$

2.5.1 $y'_i = \lfloor \frac{d'_i}{x_i} \rfloor$.

2.5.2 Se $y'_i > 0$ e $l_i x_i \geq \frac{L}{2}$ e $l_i y'_i \geq \frac{L}{2}$

2.5.2.1 $l'_n = l'_{n+1} = l_i \lceil \frac{x_i}{3} \rceil$, $a'_n = a'_{n+1} = l_i \lceil \frac{y'_i}{2} \rceil$, $n = n + 2$.

2.5.2.2 $l'_n = l'_{n+1} = l_i \lceil \frac{x_i}{3} \rceil$, $a'_n = a'_{n+1} = l_i \lfloor \frac{y'_i}{2} \rfloor$, $n = n + 2$.

2.5.2.3 Se $x_i - 2\lceil \frac{x_i}{3} \rceil > 0$

2.5.2.3.1 $l'_n = l'_{n+1} = l_i(x_i - 2\lceil \frac{x_i}{3} \rceil)$, $a'_n = l_i \lceil \frac{y'_i}{2} \rceil$, $a'_{n+1} = l_i \lfloor \frac{y'_i}{2} \rfloor$, $n = n + 2$.

2.5.3 Se $y'_i > 0$ e $l_i x_i \geq \frac{L}{2}$ e $l_i y'_i < \frac{L}{2}$

2.5.3.1 $l'_n = l'_{n+1} = l_i \lceil \frac{x_i}{3} \rceil$, $a'_n = a'_{n+1} = l_i$, $n = n + 2$.

2.5.3.2 Se $x_i - 2\lceil \frac{x_i}{3} \rceil > 0$

2.5.3.2.1 $l'_n = l_i(x_i - 2\lceil \frac{x_i}{3} \rceil)$, $a'_n = l_i$, $n = n + 1$.

2.5.4 Se $x_i y'_i < d'_i$

2.5.4.1 Se $l_i(d'_i - x_i y'_i) \geq \frac{L}{2}$

2.5.4.1.1 $l'_n = l'_{n+1} = l_i \lceil \frac{d'_i - x_i y'_i}{3} \rceil$, $a'_n = a'_{n+1} = l_i$, $n = n + 2$.

2.5.4.1.2 Se $x_i - 2\lceil \frac{d'_i - x_i y'_i}{3} \rceil > 0$

2.5.4.1.2.1 $l'_n = l_i(x_i - 2\lceil \frac{d'_i - x_i y'_i}{3} \rceil)$, $a'_n = l_i$, $n = n + 1$.

2.5.4.2 Se $l_i(d'_i - x_i y'_i) < \frac{L}{2}$

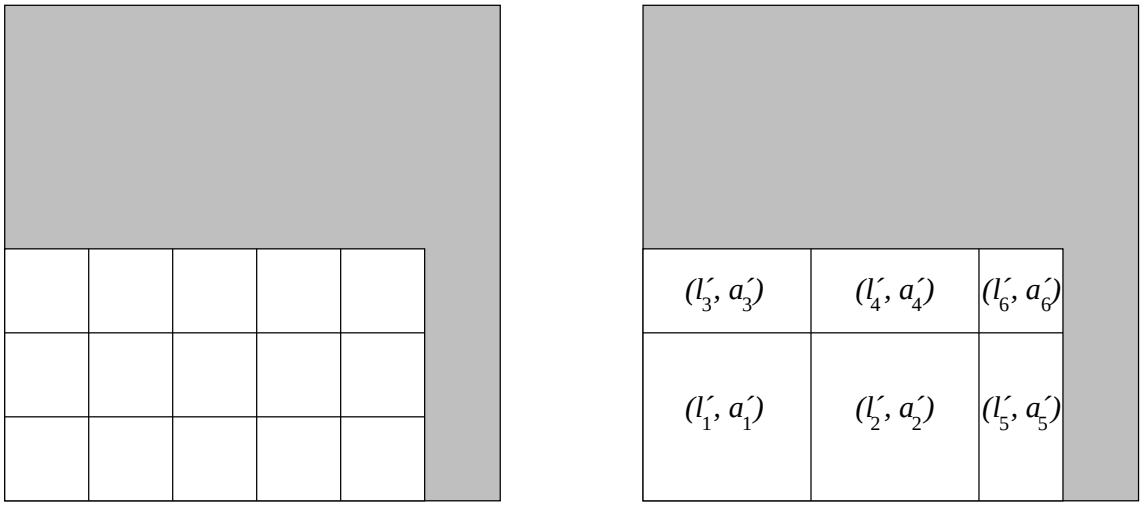
2.5.4.2.1 $l'_n = l_i(d'_i - x_i y'_i)$, $a'_n = l_i$, $n = n + 1$.

3 $n = n - 1$.

4 Se $n > 0$, resolva a instância $I' = (L, L, l' = (l'_1, \ldots, l'_n), a' = (a'_1, \ldots, a'_n))$ com o algoritmo HFF.

5 Devolva o empacotamento.

Os itens da instância I com lado maior que $\frac{L}{2}$ têm sua demanda atendida no passo 2.2. Portanto, todos os itens de I que dão origem a itens de I' têm lado no máximo $\frac{L}{2}$. Este fato juntamente com o seguinte lema garante que todo item de I' possui largura e altura



(a) $L = 59$, $l_1 = 10$, $d_1' = 15$ (b) Itens correspondentes em I'

Figura 5.4: Calculando itens de I'

no máximo $\frac{L}{2}$.

Lema 5.4.1 *Se o passo 2.5 do algoritmo PEQDSH é executado para um determinado i, então $l_i\lceil\frac{x_i}{3}\rceil \leq \frac{L}{2}$, $l_i\lceil\frac{d_i'-x_iy_i'}{3}\rceil \leq \frac{L}{2}$ e $l_i\lceil\frac{y_i'}{2}\rceil \leq \frac{L}{2}$.*

Prova. Se $x_i \leq 3$, claramente $l_i\lceil\frac{x_i}{3}\rceil \leq \frac{L}{2}$. Suponha então $x_i \geq 4$. Como $l_ix_i \leq L$, temos que $l_i \leq \frac{L}{4}$. Disto decorre que

$$l_i\left\lceil\frac{x_i}{3}\right\rceil \leq l_i\left(\frac{x_i+2}{3}\right) = \frac{l_ix_i}{3} + \frac{2l_i}{3} \leq \frac{L}{3} + \frac{L}{6} = \frac{L}{2}.$$

Como $d_i' - x_iy_i' < x_i$, temos que $l_i(d_i' - x_iy_i') < L$. Repetindo o argumento do parágrafo anterior, concluímos que $l_i\lceil\frac{d_i'-x_iy_i'}{3}\rceil \leq \frac{L}{2}$.

Finalmente, se $y_i' \leq 2$, então $l_i\lceil\frac{y_i'}{2}\rceil \leq \frac{L}{2}$. Suponha então $y_i' \geq 3$. Como $l_iy_i' \leq L$, temos que $l_i \leq \frac{L}{3}$. Além disso, $l_ix_i > L - l_i \geq L - \frac{L}{3} = \frac{2}{3}L$. Consequentemente, $l_iy_i' < \frac{2}{3}L$; caso contrário teríamos $l_i^2d_i' \geq l_ix_il_iy_i' > \frac{2}{3}L\frac{2}{3}L = \frac{4}{9}L^2$, e portanto as d_i' copias do item i teriam sido todas empacotadas no passo 2.4, e o passo 2.5 não seria executado. Dessa forma,

$$l_i\left\lceil\frac{y_i'}{2}\right\rceil \leq l_i\left(\frac{y_i'+1}{2}\right) = \frac{l_iy_i'}{2} + \frac{l_i}{2} < \frac{L}{3} + \frac{L}{6} = \frac{L}{2}.$$

■

Analisaremos a seguir o desempenho assintótico do PEQDSH.

Teorema 5.4.2 $2,4166 \approx \frac{29}{12} \leq R^{\infty}_{\text{PEQDSH}} \leq \frac{43}{16} = 2,6875$.

Prova. Mostraremos, inicialmente, que $\text{PEQDSH}(I) \leq \frac{43}{16}\ \text{OPT}(I) + 3$, para toda instância I do PEQD. Em seguida, exibiremos uma família de instâncias do PEQD tal que, para toda instância I desta família, $\text{PEQDSH}(I) = \frac{29}{12}\ \text{OPT}(I)$.

Dada uma instância I qualquer do PEQD, seja $\text{PEQDSH}(I) = k$. Seja q a quantidade de placas utilizadas no passo 2.2 contendo apenas um item. Observe que os itens contidos nestas q placas têm lado maior que $\frac{L}{2}$. Consequentemente a área aproveitada nestas q placas é maior que $\frac{L^2}{4}$. Note ainda que $\text{OPT}(I) \geq q$, pois não é possível colocar dois itens com lado maior que $\frac{L}{2}$ numa mesma placa.

Claramente, em todas as demais placas geradas no passo 2.2, a área aproveitada é maior que $\frac{4}{9}L^2$. Observe ainda que nas placas utilizadas no passo 2.4, a área aproveitada é pelo menos $\frac{4}{9}L^2$. Analisaremos agora dois casos.

Caso 1: O HFF não é utilizado na resolução da instância I. Neste caso, $\text{OPT}(I) \geq \frac{q}{4} + \frac{4}{9}(k-q)$ e portanto $\text{OPT}(I) \geq max\{q, \frac{q}{4} + \frac{4}{9}(k-q)\}$. Suponha que $q \leq \frac{q}{4} + \frac{4}{9}(k-q)$. Disto decorre que $q \leq \frac{16}{43}k$. Portanto,

$$\text{OPT}(I) \geq \frac{q}{4} + \frac{4}{9}(k-q) = \frac{4k}{9} - \frac{7q}{36} \geq \frac{4k}{9} - \frac{7}{36}\frac{16}{43}k = \frac{16k}{43} = \frac{16}{43}\text{PEQDSH}(I).$$

Suponha agora que $q > \frac{q}{4} + \frac{4}{9}(k-q)$. Disto decorre que $q > \frac{16}{43}k$. Assim,

$$\text{OPT}(I) \geq q > \frac{16k}{43} = \frac{16}{43}\text{PEQDSH}(I).$$

Caso 2: O HFF é utilizado na resolução da instância I. Observe que $n \leq 9m$, e portanto o HFF aplicado à instância I' requer tempo polinomial em m. Seja $\text{HFF}(I') = r$.

Vamos considerar os níveis produzidos no HFF indexados de 1 a t, conforme a ordem em que foram criados. Seja h_i a altura do nível i, A_i a área aproveitada no nível i, b_i a largura do nível i e c_i e largura do primeiro item de I' contido no nível i (o item mais à esquerda). Observe que, para $i < t$, $A_i \geq h_{i+1}b_i$.

Suponha que existam níveis com largura menor $\frac{2}{3}L$. Seja k o menor número tal que $b_k < \frac{2}{3}L$. Em todos os níveis seguintes os itens terão largura maior que $\frac{L}{3}$. Como todos os itens têm largura menor ou igual a $\frac{L}{2}$, em todos os níveis, exceto possivelmente no último,

teremos pelo menos 2 itens. Concluímos então que em todos os níveis, exceto no nível k e possivelmente no nível t, a largura será maior que $\frac{2}{3}L$.

Vamos considerar as placas utilizadas pelo HFF indexadas de 1 a r, conforme a ordem em que começaram a ser aproveitadas. Seja s_j a soma das alturas dos níveis contidos na placa j. Note que, para $j < r$, temos que $s_j > \frac{2}{3}L$. Seja $\mathcal{I}$ o conjunto dos índices das placas que não contêm os níveis 1, $k+1$ e t. Note que $|\mathcal{I}| \geq r-3$. Temos então que

$$\sum_{i=1}^{n} l'_i a'_i = \sum_{i=1}^{t} A_i = A_1 + \ldots + A_t > h_2 b_1 + \ldots + h_k b_{k-1} + h_{k+2} b_{k+1} + \ldots + h_t b_{t-1}$$

$$> \frac{2}{3}L(h_2 + \ldots + h_k + h_{k+2} + \ldots + h_t) \geq \frac{2}{3}L \sum_{j \in \mathcal{I}} s_j > \frac{4}{9}L^2(r-3).$$

Portanto, a área aproveitada nas placas utilizadas pelo HFF é maior que $\frac{4}{9}L^2(r-3)$. Concluímos então que $\mathrm{OPT}(I) \geq max\{q, \frac{q}{4} + \frac{4}{9}(k-q-3)\}$. Suponha agora que $q \leq \frac{q}{4} + \frac{4}{9}(k-q-3)$. Disto decorre que $q \leq \frac{16}{43}k - \frac{4}{3}$. Portanto,

$$\mathrm{OPT}(I) \geq \frac{q}{4} + \frac{4}{9}(k-q-3) = \frac{4k}{9} - \frac{7q}{36} - \frac{4}{3} \geq \frac{16k}{43} - \frac{29}{27} > \frac{16n}{43} - \frac{48}{43}.$$

Suponha que $q > \frac{q}{4} + \frac{4}{9}(k-q-3)$. Disto decorre que $q > \frac{16}{43}k - \frac{48}{43}$. Assim,

$$\mathrm{OPT}(I) \geq q > \frac{16k}{43} - \frac{48}{43}.$$

Em ambos os casos, concluímos que $\mathrm{OPT}(I) > \frac{16k}{43} - \frac{48}{43}$, donde segue que $\mathrm{PEQDSH}(I) \leq \frac{43}{16}\,\mathrm{OPT}(I) + 3$.

Vejamos agora uma família de instâncias que comprova que $R^{\infty}_{\mathrm{PEQDSH}} \geq \frac{29}{12}$. Cada instância dessa família possui $4m+1$ itens (m múltiplo de 12), divididos em 5 categorias:

- Categoria A: m itens, onde cada item $4i+1$ tem $l_{4i+1} = \frac{L}{3.2^i}$ e $d_{4i+1} = 2^{2i+2}$ $(i = 0, \ldots, m-1)$.
- Categoria B: m itens, onde cada item $4i+2$ tem $l_{4i+2} = \frac{L}{6} - 2i\epsilon$ e $d_{4i+2} = 4$ $(i = 0, \ldots, m-1)$.
- Categoria C: m itens, onde cada item $4i+3$ tem $l_{4i+3} = \frac{L}{6} - (2i+1)\epsilon$ e $d_{4i+3} = 4$ $(i = 0, \ldots, m-1)$.

- Categoria D: m itens, onde cada item $4i+4$ tem $l_{4i+4} = \frac{L}{6} - (2m+i)\epsilon$ e $d_{4i+4} = 3$ $(i = 0, \ldots, m-1)$.
- Categoria E: o item $4m+1$, com $l_{4m+1} = \frac{L}{2} + \epsilon$ e $d_{4m+1} = m$.

Claramente não é possível empacotar todos os itens com menos do que m placas, visto que não podemos colocar duas cópias do item $4m+1$ numa mesma placa. No entanto, é possível empacotar todos os itens utilizando exatamente m placas. Na Figura 5.5b mostramos que numa placa é possível empacotar todas as cópias de um item de cada uma das categorias A, B, C e D e mais uma cópia do item $4m+1$.

Na Figura 5.5a exibimos os 4 tipos de padrões utilizados pelo PEQDSH. Se $\epsilon < \frac{L}{126m-330}$, na solução produzida pelo algoritmo PEQDSH, serão utilizadas m placas (do padrão tipo I) para o item $4m+1$; 1 placa (do padrão tipo II) para cada item da categoria A; 2 placas (do padrão tipo III) para cada 6 itens das categorias B e C; e 1 placa (do padrão tipo IV) para cada 12 itens da categoria D. Portanto, o algoritmo PEQDSH utiliza $\frac{29}{12}m$ placas.

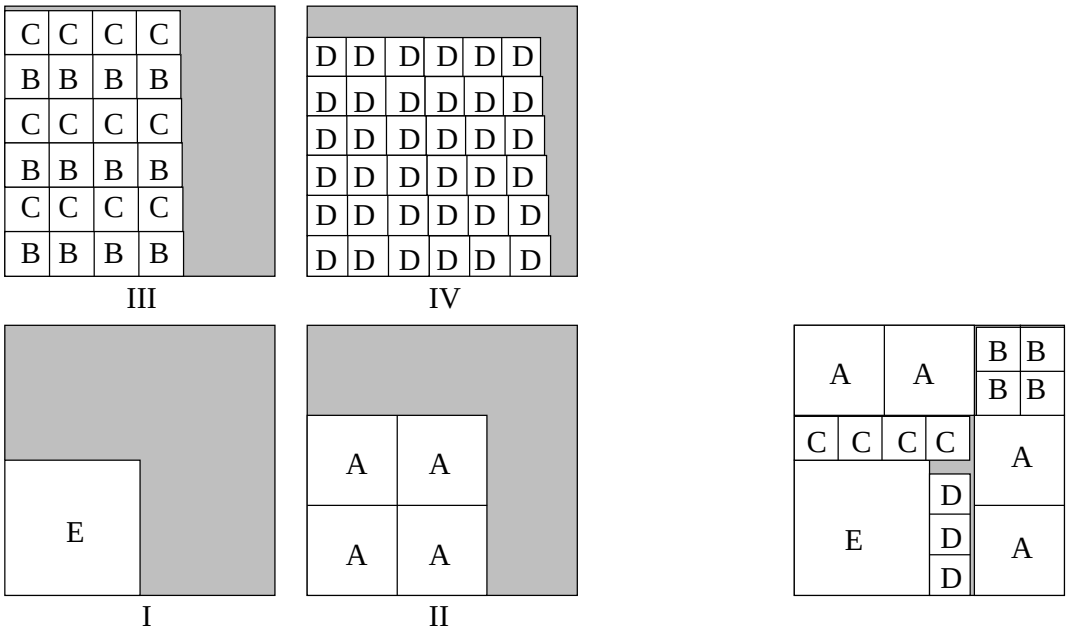
Os padrões do tipo I contêm 1 cópia do item $4m+1$ e são utilizados no passo 2.2 do algoritmo. Os padrões do tipo II contêm todas as cópias de um item da categoria A e são utilizados no passo 2.4. Os padrões do tipo III contêm todas as cópias de 3 itens da categoria B e 3 itens da categoria C e são utilizados no HFF. Finalmente, os padrões do tipo IV contêm todas as cópias de 12 itens da categoria D, que são os que têm menor altura e portanto os últimos a ser empacotados pelo HFF.

■

5.5 Empacotamentos com Rotações

Na Seção 4.4 citamos o algoritmo $\mathrm{BI}_{k,\epsilon}$, desenvolvido por Miyazawa para o PCEB_2^r, que pode ter razão de aproximação assintótica tão próxima de 2,639 quanto se queira. Tal algoritmo produz somente padrões guilhotináveis, portanto pode ser aplicado ao PCGB_2^r. No entanto, sua implementação não é trivial. Decidimos então propor e utilizar um algoritmo bastante simples, baseado no FFDH, que chamamos de *First Fit Decreasing Height using Rotations* (FFDHR).

Para que uma instância do PCGB_2^r tenha solução, é necessário que $l_i \leq L$ e $a_i \leq A$, ou $l_i \leq A$ e $a_i \leq L$ $(i = 1, \ldots, m)$. No entanto, sem perda de generalidade, vamos supor que



(a) Padrões utilizados pelo PEQDSH (b) Padrão numa solução ótima

Figura 5.5: Parte de uma solução do encontrada pelo PEQDSH.

todos os itens têm altura menor ou igual a A e largura menor ou igual a L (se necessário, fazemos com que os itens sofram rotação).

O FFDHR utiliza a estratégia do algoritmo FFDH, com duas modificações. Ao empacotar o item i, procuramos o nível de menor índice no qual ele caiba, ou na sua orientação original, ou após ter sofrido uma rotação. Se existir um tal nível, o item i é empacotado nesse nível, dando prioridade para o uso da orientação original. Se não existir um tal nível, é criado um novo nível, onde esse item é empacotado na sua orientação original.

A outra modificação é que os níveis (a partir de 1) são criados nas placas e não numa faixa de altura ilimitada. Vamos considerar que as placas estão indexadas a partir de 1. Suponha que ao empacotar um item i, haja necessidade de criar um novo nível. Neste caso, procuramos a placa de menor índice na qual é possível criar um novo nível onde caiba o item i na sua orientação original. Observe que somente realizamos uma rotação num item se for para evitar a criação de um novo nível. A seguir descrevemos detalhadamente o FFDHR.

Na descrição do FFDHR, $\mathcal{N}$ é o índice do último nível criado, $\mathcal{P}$ é o índice da última placa utilizada e t_i representa a posição do eixo y correspondente ao topo do último nível criado na placa i. Além disso, w_j e h_j representam a largura da parte ainda não ocupada

Algoritmo 5.4 FFDHR

Entrada: Uma instância $I = (L, A, l, a)$ do PCEB$_2$, onde $l = (l_1, \ldots, l_m)$ e $a = (a_1, \ldots, a_m)$.

Saída: Uma solução para I.

Ordene os itens em ordem decrescente de altura, obtendo $a_1 \geq \ldots \geq a_m$.
Faça $\mathcal{N} = 0$, $\mathcal{P} = 0$.
Para $i = 1$ até m
  $d = l_i$.
  $k = min(\{j \mid 1 \leq j \leq \mathcal{N}$ e $(l_i \leq w_j$ ou $(a_i \leq w_j$ e $l_i \leq h_j))\} \cup \{\mathcal{N}+1\})$.
  Se $k \leq \mathcal{N}$ e $l_i > w_k$ faça $d = a_i$. `/* Item deve sofrer rotação */`
  Se $k > \mathcal{N}$ `/* É preciso criar um novo nível */`
    $r = min(\{j \mid 1 \leq j \leq \mathcal{P}$ e $t_j + a_i \leq A\} \cup \{\mathcal{P}+1\})$.
    $p_k = r$, $w_k = L$.
    Se $r > \mathcal{P}$ `/* É preciso utilizar uma nova placa */`
      $\mathcal{P} = \mathcal{P} + 1$ e $t_r = 0$.
    $h_k = l_i$, $b_k = t_r$ e $t_r = t_r + l_i$.
  Se $d = a_i$
    Empacote o item i na posição $(L - w_k, b_k)$ da placa p_k, após o item sofrer rotação.
  Senão
    Empacote o item i na posição $(L - w_k, b_k)$ da placa p_k.
  $w_k = w_k - d$.
Devolva o empacotamento.

e a altura do nível j, respectivamente, b_j é a posição do eixo y correspondente à base do nível j, e p_j indica qual é a placa onde foi criado o nível j. A variável d é utilizada para guardar a dimensão na horizontal que o item i vai ocupar no nível (l_i se o item for empacotado na sua orientação original ou a_i, caso contrário).

Sobre a razão de aproximação assintótica do FFDHR, temos o seguinte resultado.

Teorema 5.5.1 FFDHR$(I) \leq 4$ OPT$(I) + 2$, *para toda instância* I *do* $PCGB_2^r$.

Prova. Ao final da execução do algoritmo FFDHR, temos que $\mathcal{N}$ é a quantidade de níveis criados, $\mathcal{P}$ é a quantidade de placas utilizadas e t_i é a posição do eixo y correspondente ao topo do último nível criado na placa i, ou seja, a soma das altura dos níveis contidos na placa i. Além disso, w_j e h_j indicam a largura da parte não ocupada e a altura do nível j, respectivamente. Como o primeiro item empacotado em cada nível não sofre rotação e

os itens foram colocados em ordem decrescente de altura, temos que $h_1 \geq \ldots \geq h_{\mathcal{N}}$.

Vamos chamar de itens *grandes* aqueles que têm altura maior que $\frac{A}{2}$ e largura maior que $\frac{L}{2}$. Chamemos as placas que contêm itens grandes de placas *iniciais*. Observe que a área de cada item grande é maior que $\frac{LA}{4}$ e portanto a área aproveitada nas placas iniciais é maior que $\frac{LA}{4}$.

Chamemos de itens *altos* aqueles que não são itens grandes mas que têm altura maior que $\frac{A}{2}$. Analisemos agora as placas que não contêm itens grandes mas que contêm itens altos e que têm índice menor que $\mathcal{P}$. Chamemos tais placas de *intermediárias*. As placas que não contêm itens grandes nem altos e que têm índice menor que $\mathcal{P}$ serão chamadas de placas *finais* e denotaremos por q a quantidade de tais placas.

Note que as placas intermediárias, com exceção possivelmente da última, possuem pelo menos um nível contendo itens altos, sendo que a soma das larguras destes itens altos é maior que $\frac{L}{2}$. Sendo assim, em tais placas intermediárias a área aproveitada é maior que $\frac{LA}{4}$.

Finalmente, vamos analisar as placas finais. Observe que toda placa final i possui $t_i > \frac{2}{3}A$. Note também que se um item i sofreu rotação antes de ser empacotado num nível j, então naquele instante $l_i > w_j$. Como $w_j \leq a_i$ (senão o item i não poderia ser empacotado no nível j, mesmo sofrendo rotação), temos que $l_i > a_i$. Sendo assim, é válido que a altura (na orientação em que foi empacotado) de todo item contido num nível $j < \mathcal{N}$ é maior ou igual a h_{j+1}.

Para cada nível j vamos denotar por c_j a largura do primeiro item (aquele mais à esquerda) empacotado nesse nível j. Observe que, para $i < j \leq \mathcal{N}$, $b_i + c_j > L$ e $h_i \geq h_j$. Assim, se para cada nível i denotarmos por A_i a área aproveitada nesse nível, temos que para $i < j \leq \mathcal{N}$,

$$A_i + A_j \geq h_j b_i + h_j c_j = h_j(b_i + c_j) > h_j L.$$

Dessa forma, denotando por $\mathcal{F}$ o conjunto dos índices dos níveis contidos nas placas finais e sendo k o nível de menor índice contido numa placa final, temos que

$$2\sum_{i\in\mathcal{F}} A_i > L \sum_{i\in\mathcal{F}-\{k\}} h_i > \frac{2}{3}LA(q-1).$$

Concluímos então que a área aproveitada em $q-1$ placas finais é maior que $\frac{LA}{3}$. Sendo assim,

$$\mathrm{OPT}(I) \geq \frac{\sum_{i=1}^{\mathcal{N}} A_i}{LA} > \frac{LA(\mathcal{P}-3)}{4LA} = \frac{\mathcal{P}-3}{4} = \frac{\mathrm{FFDHR}(I)-3}{4}.$$

Como $\mathrm{FFDHR}(I)$ e $\mathrm{OPT}(I)$ são inteiros, temos que $\mathrm{FFDHR}(I) \leq 4\ \mathrm{OPT}(I) + 2$. ■

Não conseguimos mostrar que a razão de aproximação assintótica expressa no Teorema 5.5.1 é justa. É possível que, utilizando técnicas mais sofisticadas de análise, esta razão de aproximação possa ser melhorada.

Capítulo 6

Programação Dinâmica para o $PCGV_2$

6.1 Introdução

O problema de corte de guilhotina bidimensional com valor ($PCGV_2$) consiste em: dada uma placa, de largura L e altura A, e uma lista de m itens, cada item i com largura $l_i \leq L$, altura $a_i \leq A$ e valor v_i, queremos determinar como cortar a placa, fazendo apenas cortes de guilhotina, de modo a maximizar a soma dos valores dos itens produzidos. Convém salientar que ao cortar a placa é possível produzir diversas cópias de um mesmo item.

Neste capítulo, vamos considerar que L, A, l_1, ..., l_m, a_1, ..., a_m são inteiros. A partir de uma instância do $PCGV_2$ onde tais dados são números racionais, podemos obter uma instância equivalente onde estes dados são números inteiros, simplesmente mudando a escala usada para medir as dimensões da placa e dos itens. Fazemos isto multiplicando as larguras (da placa e dos itens) por um número λ_1 apropriado e as alturas (da placa e dos itens) por um número λ_2 apropriado[1].

Além disso, vamos supor que rotações não são permitidas (se necessário, replicamos cada item como explicado na Subseção 3.3.5). Veremos a seguir que o $PCGV_2$ pode ser resolvido através de programação dinâmica.

[1]Representando as larguras através de frações de números inteiros, um valor apropriado para λ_1 é o *mínimo múltiplo comum* dos denominadores destas frações. De forma análoga podemos determinar um valor apropriado para λ_2.

6.2 Aplicabilidade da Programação Dinâmica ao PCGV$_2$

Dada uma instância I de um problema, chamamos de *decomposição* de I o resultado da subdivisão de I em diversas instâncias menores (segundo alguma métrica) e que sejam do mesmo tipo que I. A programação dinâmica é uma generalização da bem conhecida técnica de *dividir-e-conquistar* e consiste basicamente em: dada uma instância I, decompor esta instância de diversas maneiras e, para cada decomposição, calcular soluções ótimas das instâncias que constituem a decomposição e, a partir destas soluções ótimas parciais, encontrar uma solução ótima para I. Esta técnica de resolução de problemas é especialmente bem sucedida se a quantidade de decomposições que tivermos que examinar for pequena.

Alguns problemas têm propriedades que permitem que eles sejam resolvidos através da técnica de programação dinâmica. Por exemplo, considere um problema $\mathcal{P}$ de maximização ou de minimização cujas instâncias podem ser decompostas em instâncias de $\mathcal{P}$. Podemos resolver $\mathcal{P}$ usando programação dinâmica se, para toda instância $I \in \mathcal{P}$, o valor de uma solução ótima de I for igual à soma dos valores de soluções ótimas das instâncias obtidas através de alguma decomposição de I.

Diversos problemas possuem a propriedade citada no parágrafo anterior, tais como o problema da subcadeia comum máxima [CLRS01], o problema de alinhamento de sequências [BMR95], o problema da mochila inteira [CLRS01], etc. O PCGV$_2$ também possui tal propriedade, embora o número de decomposições que precisamos examinar nem sempre seja pequeno.

O primeiro algoritmo para o PCGV$_2$, baseado em programação dinâmica, foi proposto por Gilmore e Gomory [GG65]. Sem exagero, pode-se dizer que os artigos escritos por estes autores representaram um marco importante no estudo de problemas de corte e empacotamento, sendo que um grande número de algoritmos exatos, algoritmos de aproximação e heurísticas para tais problemas são baseados nas idéias propostas por Gilmore e Gomory. Segundo o sítio *NEC Research Institute CiteSeer*[2], tais artigos foram citados mais de uma centena de vezes nos artigos catalogados em sua base de dados, até a data em que esta tese foi finalizada. Os artigos de Gilmore e Gomory aparecem nas referências bibliográficas deste texto. Apresentamos a seguir a estratégia proposta por Gilmore e Gomory para resolver o PCGV$_2$.

[2] `http://citeseer.nj.nec.com`

6.3 As Fórmulas de Recorrência de Gilmore e Gomory

Gilmore e Gomory [GG65] desenvolveram fórmulas de recorrência, que podem ser calculadas através de programação dinâmica, para resolver o $PCGV_2$. Tais fórmulas são baseadas no fato de que um padrão guilhotinável é obtido através de uma sequência de estágios de corte (veja Subseção 3.2.1.6). Herz [Her72] observou que havia um erro nas fórmulas de recorrências propostas por Gilmore e Gomory, e mostrou como corrigir tal erro. Anos mais tarde, Beasley [Bea85a] apontou outro erro nas fórmulas de recorrência, indicando como torná-las corretas. Apresentaremos aqui as fórmulas de recorrência de Gilmore e Gomory com as devidas correções.

Para entender a fórmula de recorrência precisaremos de algumas notações e definições. Chamaremos de *valor de um padrão* a soma dos valores dos itens contidos no padrão. Denotaremos por $F(k, l, a)$ o valor ótimo de um padrão guilhotinável numa placa de dimensões $(l,\ a)$ obtido através de no máximo k estágios de cortes, sendo que no primeiro estágio os cortes são paralelos ao eixo horizontal. De forma similar, $G(k, l, a)$ representará o valor ótimo de um padrão guilhotinável numa placa de dimensões $(l,\ a)$ obtido através de no máximo k estágios de cortes, sendo que no primeiro estágio os cortes são paralelos ao eixo vertical.

Sejam $\mathcal{A} = \{1, \ldots, \lfloor \frac{a}{2} \rfloor\}$ e $\mathcal{L} = \{1, \ldots, \lfloor \frac{l}{2} \rfloor\}$. Podemos calcular $F(k, l, a)$ e $G(k, l, a)$ utilizando as seguintes fórmulas de recorrência:

$$F(k, l, a) = max(F(0, l, a),\ \{F(k, l, a') + F(k, l, a - a') \mid a' \in \mathcal{A}\},\ G(k - 1, l, a)) \quad (6.1)$$

$$G(k, l, a) = max(G(0, l, a),\ \{G(k, l', a) + F(k, l - l', a) \mid l' \in \mathcal{L}\},\ F(k - 1, l, a)). \quad (6.2)$$

Estas recorrências têm como base os termos de F e de G em que k é igual a 0 (zero). Os termos $F(0, l, a)$ e $G(0, l, a)$ representam o maior valor que pode ser obtido numa placa de dimensões $(l,\ a)$ sem que se faça nenhum estágio de corte, ou seja, numa placa contendo apenas um item (eventualmente serão necessários dois cortes com o propósito de aparar as sobras). Dessa forma,

$$F(0, l, a) = G(0, l, a) = max(\{v_i |\ l_i \leq l,\ a_i \leq a, 1 \leq i \leq m\} \cup \{0\}).$$

Dada uma instância I do PCGV$_2$, o valor ótimo de um padrão guilhotinável numa placa de dimensões (L, A), obtido através de no máximo k estágios de cortes, é dado por $max(F(k, L, A),\ G(k, L, A))$.

Note que o tempo requerido para resolver (6.1) e (6.2) depende de l e de a. Dessa forma, dada uma instância do PCGV$_2$ onde as dimensões são fracionárias, se fizermos a mudança de escala explicada no início deste capítulo, a computação de (6.1) e (6.2) pode se tornar inviável, em termos práticos. Sendo assim, a suposição de que as dimensões da placa e dos itens são dadas por números inteiros constitui-se numa severa restrição para a aplicação destas fórmulas de recorrência.

A abordagem proposta por Gilmore e Gomory, discutida nesta seção, é especialmente adequada quando o processo de corte limita a quantidade de estágios. Apresentaremos a seguir a fórmula de recorrência proposta por Beasley, na qual a quantidade de estágios não é limitada.

6.4 A Fórmula de Recorrência de Beasley

Num padrão guilhotinável, cada corte secciona um retângulo em 2 retângulos. Assim, se r é a quantidade máxima de itens que podem estar contidos num padrão, podemos ter padrões que necessitam de $r-1$ cortes e eventualmente $r-1$ estágios de corte (supondo 1 corte em cada estágio). Observe ainda que r pode ser exponencial em m, se os itens forem muito pequenos em comparação com a placa.

Nos processos de corte de guilhotina em que a quantidade de estágios não é limitada, podemos garantir que as fórmulas (6.1) e (6.2) produzem a resposta correta se utilizarmos k igual a r. Obviamente, o cálculo destas fórmulas pode se tornar impraticável se r for exponencial em m. Para tais processos de corte, pode ser vantajoso utilizar a fórmula de recorrência proposta por Beasley, que não é baseada em estágios de corte.

Para descrevê-la, é conveniente adotar mais algumas notações. Denotaremos por $v(l, a)$ o valor de um item mais valioso que cabe numa placa de dimensões (l, a), ou 0 se nenhum item cabe na placa[3]. Mais formalmente,

$$v(l, a) = max(\{v_i \mid 1 \leq i \leq m,\ \ l_i \leq l,\ a_i \leq a\} \cup \{0\}).$$

[3]Note que $v(l, a)$ é equivalente a $F(0, l, a)$.

O resultado que se segue nos permite deduzir a fórmula de recorrência proposta por Beasley.

Proposição 6.4.1 *Seja* $I = (L,\ A,\ l,\ a,\ v)$ *uma instância do* $PCGV_2$ *onde* $l = \{l_1,\ \ldots,\ l_m\}$, $a = \{a_1,\ \ldots,\ a_m\}$ *e* $v = \{v_1,\ \ldots,\ v_m\}$. *Sejam* $I_1 = (L',\ A,\ l,\ a,\ v)$, $I_2 = (L\ -\ L',\ A,\ l,\ a,\ v)$, $I_3 = (L,\ A',\ l,\ a,\ v)$ *e* $I_4 = (L,\ A\ -\ A',\ l,\ a,\ v)$. *Para algum* $1 \leq L' \leq \lfloor \frac{L}{2} \rfloor$ *e algum* $1 \leq A' \leq \lfloor \frac{A}{2} \rfloor$ *(*$L', A' \in \mathbb{N}$*) temos que:*

$$\mathrm{OPT}(I) = max(v(L,A),\ \ \mathrm{OPT}(I_1) + \ \mathrm{OPT}(I_2),\ \ \mathrm{OPT}(I_3) + \ \mathrm{OPT}(I_4)).$$

Prova. Seja $\mathcal{P}$ um padrão que corresponde a uma solução ótima de I. Se não existe nenhum corte de guilhotina em $\mathcal{P}$, claramente $\mathrm{OPT}(I) = v(L,A)$.

Suponha que o primeiro corte de guilhotina é feito na vertical (como na Figura 3.1b), seccionando a placa em duas placas menores (subplacas) de dimensões $(L',\ A)$ e $(L - L',\ A)$. Note que podemos considerar que $L' \leq \lfloor \frac{L}{2} \rfloor$, caso contrário trocamos L' por $L - L'$. Claramente os valores das soluções obtidas nestas duas subplacas são ótimos, caso contrário $\mathcal{P}$ não seria ótimo. Dessa forma, $\mathrm{OPT}(I) = \ \mathrm{OPT}(I_1) + \ \mathrm{OPT}(I_2)$.

Suponha agora que o primeiro corte de guilhotina é feito na horizontal, seccionando a placa em duas subplacas de dimensões $(L,\ A')$ e $(L,\ A - A')$. Usando um raciocínio análogo ao anterior, concluímos que $\mathrm{OPT}(I) = \ \mathrm{OPT}(I_3) + \ \mathrm{OPT}(I_4)$. ■

Vamos denotar por $V(l,a)$ o valor ótimo de um padrão guilhotinável numa placa de dimensões $(l,\ a)$. Novamente, sejam $\mathcal{A} = \{1, \ldots, \lfloor \frac{a}{2} \rfloor\}$ e $\mathcal{L} = \{1, \ldots, \lfloor \frac{l}{2} \rfloor\}$. Podemos calcular $V(l,a)$ através da seguinte fórmula de recorrência:

$$V(l,a) = max(v(l,a), \{V(l',a)+V(l-l',a) | l' \in \mathcal{L}\}, \{V(l,a')+V(l,a-a') | a' \in \mathcal{A}\}). \quad (6.3)$$

Ao calcular um padrão ótimo $\mathcal{P}$ para uma placa de dimensões $(l,\ a)$, temos três possibilidades:

- Nenhum corte de guilhotina é feito em $\mathcal{P}$. Neste caso, $V(l,a) = v(l,a)$.
- O primeiro corte de guilhotina em $\mathcal{P}$ é feito numa posição l' do eixo horizontal. Dessa forma, $V(l,a) = V(l',a) + V(l - l',a)$.
- O primeiro corte de guilhotina em $\mathcal{P}$ é feito numa posição a' do eixo vertical. Neste caso, $V(l,a) = V(l,a') + V(l,a - a')$.

Concluímos então que a fórmula de recorrência (6.3) realmente é válida. Dessa forma, dada uma instância I do PCGV$_2$, ao calcular $V(L, A)$ obtemos o valor de uma solução ótima de I.

Assim como em (6.1) e (6.2), o tempo requerido para resolver (6.3) depende de l e de a. Beasley propôs utilizar os *pontos de discretização*, definidos por Herz, com o intuito de resolver (6.3) mais rapidamente. Tal modificação também tem como consequência o fato de que a mudança de escala que explicamos no início deste capítulo não aumenta a dificuldade de se resolver o problema. Na próxima seção discutimos como isto pode ser feito.

6.5 Os Pontos de Discretização de Herz

Herz [Her72] propôs um algoritmo recursivo para o PCGV$_2$. Dada uma instância I do PCGV$_2$, tal algoritmo implementa um procedimento que percorre uma árvore cujos ramos (um caminho desde a raiz até uma folha) representam todos os padrões guilhotináveis para a instância I. Este algoritmo utiliza o que Herz chamou de *pontos de discretização* para limitar as ramificações da árvore que serão percorridas. Podemos utilizar tais pontos para tornar a computação de (6.3) mais eficiente.

Chamamos de *ponto de discretização da largura* um valor $i \leq L$ que pode ser obtido através de uma combinação cônica inteira[4] de $l_1, \ldots, l_m$. Analogamente, chamamos de *ponto de discretização da altura* um valor $j \leq A$ que pode ser obtido através de uma combinação cônica inteira de $a_1, \ldots, a_m$.

Mais precisamente, sejam $l = (l_1, \ldots, l_m)^T$, $a = (a_1, \ldots, a_m)^T$, $P = \{l^T z \mid l^T z \leq L,\ z \in \mathbb{N}^m\}$ e $Q = \{a^T z \mid a^T z \leq A,\ z \in \mathbb{N}^m\}$. Os elementos de P e Q constituem os pontos de discretização da largura e da altura, respectivamente. Tais pontos podem ser calculados como explicados na próxima subseção.

6.5.1 Calculando os Pontos de Discretização

Discutiremos agora dois algoritmos para calcular os pontos de discretização. O primeiro é um algoritmo de enumeração explícita. O segundo é baseado em programação dinâmica.

[4]Uma combinação cônica inteira é uma combinação linear onde todos os coeficientes são números inteiros não-negativos.

6.5.1.1 Utilizando Enumeração Explícita

Podemos calcular os pontos de discretização da largura enumerando todas as combinações cônicas inteiras de $l_1, \ldots, l_m$ cujo valor é menor ou igual a L. Podemos calcular os pontos de discretização da altura de forma análoga. Tal enumeração pode ser feita utilizando-se o algoritmo DEE (discretização por enumeração explícita) que descrevemos a seguir.

Algoritmo 6.1 DEE

Entrada: D, d_1, ..., d_m.

Saída: P, o conjunto dos pontos de discretização.

1 $P = \emptyset$, $k = 0$.

2 Enquanto $k \geq 0$ faça

2.1 Para $i = k + 1$ até m faça $z_i = \left\lfloor \frac{D - \sum_{j=1}^{i-1} d_j z_j}{d_i} \right\rfloor$.

2.2 $P = P \cup \{\sum_{j=1}^{m} z_j d_j\}$.

2.3 $k = max(\{i \mid z_i > 0,\ 1 \leq i \leq m\} \cup \{-1\})$.

2.4 Se $k > 0$ faça $z_k = z_k - 1$ e $P = P \cup \{\sum_{j=1}^{k} z_j d_j\}$.

3 Devolva P.

No algoritmo DEE, D representa a largura (ou altura) da placa e d_1, ..., d_m representam as larguras (ou alturas) dos itens. O DEE pode ser implementado de forma a requerer tempo $\mathcal{O}(m\delta)$, onde δ representa a quantidade de combinações cônicas inteiras de d_1, ..., d_m com valor menor ou igual a D. Observe que $\delta \leq \lfloor \frac{D}{d_{min}} \rfloor^m$, onde $d_{min} = min(d_1, \ldots, d_m)$. Isto significa que ao multiplicar D, d_1, ..., d_m por uma constante λ, o tempo requerido pelo DEE não é afetado. Sendo assim, uma eventual mudança de escala para tornar inteiras as dimensões da placa e dos itens não tem impacto sobre a complexidade do algoritmo DEE.

Por outro lado, sabemos que $|P| \leq D$, ou seja, a quantidade de pontos de discretização é no máximo D. No entanto, δ pode ser maior que $2^{\sqrt{D}}$. Considere uma entrada para o algoritmo DEE onde $D = m^2$ e $d_i = i$ $(i = 1, \ldots, m)$. Seja $\alpha = \{d_i, \ldots, d_m\}$. Observe que a soma dos elementos de qualquer subconjunto de α é menor ou igual a D. Logo, para uma tal entrada, $\delta \geq 2^m = 2^{\sqrt{D}}$.

Observe no entanto que, se pudermos garantir que $d_i > \frac{D}{k}$ $(i = 1, \ldots, m)$, a soma dos m coeficientes de qualquer combinação cônica inteira de d_1, ..., d_m com valor menor

ou igual a D tem que ser um número entre 0 e $k-1$. Sendo assim, δ é menor ou igual à quantidade de combinações com repetição de m elementos tomados k a k. Portanto, para k fixo, δ é polinomial em m e consequentemente o algoritmo DEE requer tempo polinomial em m. Ademais, neste caso, a quantidade de pontos de discretização também será polinomial em m.

Discutiremos a seguir outro algoritmo para calcular os pontos de discretização que pode ser mais eficiente em muitas situações.

6.5.1.2 Utilizando Programação Dinâmica

Podemos utilizar programação dinâmica para calcular os pontos de discretização. A idéia básica é resolver um problema da mochila em que cada item i tem peso e valor igual a d_i $(i = 1, \ldots, m)$ e a mochila tem capacidade D. A conhecida técnica de programação dinâmica para o problema da mochila nos fornece o valor ótimo para mochilas de capacidade (inteira) variando de 1 a D. É fácil perceber que j é um ponto de discretização se e somente se a mochila de capacidade j tem valor ótimo igual a j. Temos então um algoritmo, que chamaremos de DPD (discretização usando programação dinâmica), cujos detalhes são dados a seguir.

Algoritmo 6.2 DPD

Entrada: D, d_1, ..., d_m.

Saída: P, o conjunto dos pontos de discretização.

$P = \{0\}$.
Para $j = 0$ até D faça $c_j = 0$.
Para $i = 1$ até m faça
  Para $j = d_i$ até D
    Se $c_j < c_{j-d_i} + d_i$ então $c_j = c_{j-d_i} + d_i$
Para $j = 1$ até D
  Se $c_j = j$ então $P = P \cup \{j\}$.
Devolva P.

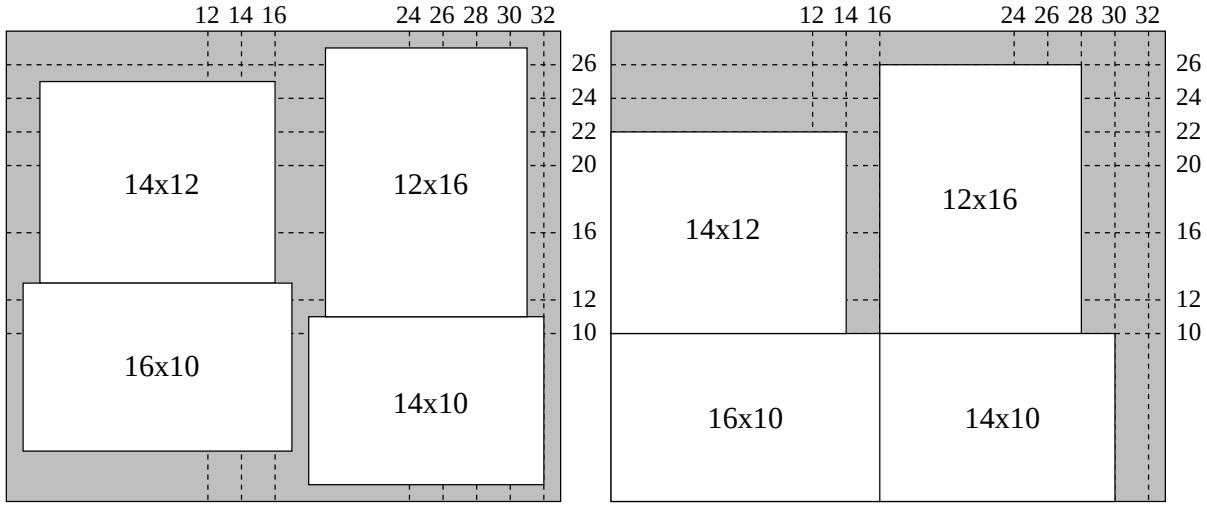
Observe que o algoritmo DPD requer tempo $\mathcal{O}(mD)$. Sendo assim, a mudança de escala com o objetivo de tornar as dimensões da placa e dos itens inteiras pode tornar a utilização do DPD inviável, em termos práticos. Por outro lado, o DPD é especialmente

adequado quando o valor de D é pequeno. Nos casos em que D é grande mas as dimensões da placa não são muito maiores que as dimensões dos itens, o algoritmo DEE possui um desempenho satisfatório. No entanto, nos testes computacionais que realizamos, apresentados nos capítulos 6, 7 e 8, utilizamos o algoritmo DPD para calcular os pontos de discretização.

Agora que sabemos o que são pontos de discretização, explicaremos o que são padrões canônicos e mostraremos que sempre existe uma solução ótima para o $PCGV_2$ que corresponde a um padrão canônico.

6.5.2 Padrões Canônicos

Herz definiu *padrão canônico* como sendo um padrão onde todos os cortes são feitos em pontos de discretização. Os padrões da Figura 3.1 e da Figura 6.1b são canônicos. Já o padrão da Figura 6.1a não é canônico, se supormos que existem apenas itens de dimensões (14, 12), (16, 10), (12, 16) e (14, 10). Observe que alguns cortes são feitas em posições que não correspondem a pontos de discretização (tais pontos são aqueles correspondentes às linhas pontilhadas).



(a) Padrão não-canônico (b) Padrão canônico

Figura 6.1: Padrões equivalentes

Veremos agora que basta considerar somente padrões canônicos na busca de uma solução ótima para o $PCGV_2$. Antes, vejamos uma definição. Dizemos que dois padrões,

para uma dada instância, são *equivalentes* se eles possuem os mesmos itens, nas mesmas quantidades.

Proposição 6.5.1 *Seja $I = (L, A, l, a, v)$ uma instância do PCGV$_2$ e $\mathcal{P}$ um padrão guilhotinável para a instância I. Então existe um padrão guilhotinável canônico para I equivalente a $\mathcal{P}$.*

Prova. Seja k o número de cortes de guilhotina requeridos pelo padrão $\mathcal{P}$. Faremos indução em k. Se $k = 0$, obviamente a proposição é válida, pois nenhum corte de guilhotina precisa ser feito, e portanto $\mathcal{P}$ é canônico.

Suponha agora $k \geq 1$. Sem perda de generalidade, suponha que o primeiro corte de guilhotina de $\mathcal{P}$ é feito na vertical, numa posição i. Sejam $\mathcal{P}_1$ e $\mathcal{P}_2$ os padrões correspondentes às subplacas obtidas ao efetuar o primeiro corte de guilhotina de $\mathcal{P}$, sendo $\mathcal{P}_1$ o padrão correspondente à subplaca da esquerda. Se i é um ponto de discretização, aplicamos a hipótese de indução em $\mathcal{P}_1$ e $\mathcal{P}_2$, obtendo dois padrões guilhotináveis canônicos $\mathcal{P}'_1$ e $\mathcal{P}'_2$ equivalentes a $\mathcal{P}_1$ e $\mathcal{P}_2$, respectivamente. Colocando $\mathcal{P}'_1$ à esquerda de $\mathcal{P}'_2$ obtemos um padrão guilhotinável canônico equivalente a $\mathcal{P}$.

Suponha então que i não é um ponto de discretização. Seja $\mathcal{P}^*_1$ o padrão obtido movendo-se cada item contido em $\mathcal{P}_1$ para a posição mais à esquerda possível, sem que ocorra sobreposição (este procedimento é ilustrado pela Figura 6.1). Seja i^* a posição mais à direita ocupada por um item de $\mathcal{P}^*_1$. Como i^* é um ponto de discretização, obviamente $i^* < i$. Efetue em $\mathcal{P}^*_1$ um corte de guilhotina vertical, na posição i^*, obtendo duas subplacas $\mathcal{S}_1$ e $\mathcal{S}_2$, sendo $\mathcal{S}_1$ a subplaca que contém o padrão equivalente a $\mathcal{P}^*_1$. Pela hipótese de indução, existe um padrão guilhotinável canônico $\mathcal{P}'_1$ equivalente a $\mathcal{P}^*_1$ e portanto equivalente a $\mathcal{P}_1$.

Utilizando em $\mathcal{P}_2$ a mesma construção delineada no parágrafo anterior, obtemos um padrão guilhotinável canônico $\mathcal{P}'_2$ equivalente a $\mathcal{P}_2$. Colocando $\mathcal{P}'_1$ à esquerda de $\mathcal{P}'_2$, de modo que i^* corresponda ao ponto mais à esquerda de $\mathcal{P}'_2$, obtemos um padrão guilhotinável canônico equivalente a $\mathcal{P}$. ■

6.5.3 Modificando a Fórmula de Recorrência de Beasley

A Proposição 6.5.1 tem como consequência o fato de que é suficiente procurarmos uma solução para o PCGV$_2$ apenas entre os padrões canônicos. Portanto, na recorrência (6.3) somente precisamos exigir que $l' \in P$ e $a' \in Q$, onde P e Q contêm os pontos de

discretização da largura e da altura, respectivamente. Observe, no entanto, que $l - l'$ e $a - a'$ podem não ser pontos de discretização.

Definimos $p(x)$ como sendo o maior ponto de discretização da largura que é menor ou igual a x. De forma análoga, $q(y)$ representa o maior ponto de discretização da altura que é menor ou igual a y. Convém observar que 0 é um ponto de discretização da largura e da altura, portanto, para $x \geq 0$ e $y \geq 0$, o valor $p(x)$ e de $q(y)$ está bem definido. Formalmente:

$$p(x) = max(i \mid i \in P,\ i \leq x) \quad \text{e} \quad q(y) = max(j \mid j \in Q,\ j \leq y).$$

Obtemos então a seguinte fórmula de recorrência.

$$\begin{aligned} V(l,a) = max(v(l,a),\ &\{V(l',\ a) + V(p(l-l'),\ a) \mid l' \in P\}, \\ &\{V(l,\ a') + V(l,\ q(a-a')) \mid a' \in Q\}). \end{aligned} \tag{6.4}$$

Veremos a seguir um algoritmo que calcula (6.4) utilizando programação dinâmica.

6.6 O Algoritmo PCGV$_2$PD

Desenvolvemos o algoritmo PCGV$_2$PD que resolve a fórmula de recorrência proposta por Beasley. Além de resolver (6.4), o PCGV$_2$PD permite reconstruir um padrão correspondente a uma solução ótima[5]. Para isto, o algoritmo armazena numa matriz, para toda placa de largura $l' \in P$ e altura $a' \in Q$, qual a direção e posição do primeiro corte de guilhotina a ser feito na placa. No caso de nenhum corte de guilhotina ser feito numa placa, o algoritmo armazena qual item deve ser colocado nesta placa. Descrevemos a seguir os detalhes do PCGV$_2$PD.

No final do algoritmo PCGV$_2$PD, dada uma placa de dimensões $(p_i,\ q_j)$, $1 \leq i \leq r$ e $1 \leq j \leq s$, $V(i,j)$ contém o valor ótimo que pode ser obtido na placa, $guilhotina(i,j)$ indica a direção do primeiro corte de guilhotina e $posicao(i,j)$ é a posição, no eixo x ou no eixo y, onde deve ser feito o primeiro corte de guilhotina. Se $guilhotina(i,j) = nil$, então

[5]Para representar um padrão de corte bidimensional utilizaremos as mesmas convenções adotadas na Seção 4.4.

nenhum corte de guilhotina deve ser feito na placa. Neste caso, $item(i,j)$ (se diferente de zero) indica qual item deve ser colocado na placa. O valor de uma solução ótima estará em $V(r,s)$.

Algoritmo 6.3 PCGV$_2$PD

Entrada: Uma instância $I = (L, A, l, a, v)$ do PCEV$_2$, onde $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $v = (v_1, \ldots, v_m)$.

Saída: Uma solução ótima de I.

Calcule $p_1 < \ldots < p_r$, os pontos de discretização da largura L.
Calcule $q_1 < \ldots < q_s$, os pontos de discretização da altura A.
Para $i = 1$ até r
  Para $j = 1$ até s
    $V(i,j) = max(\{v_k \mid 1 \leq k \leq m,\ l_k \leq p_i \text{ e } a_k \leq q_j\} \cup \{0\})$.
    $item(i,j) = max(\{k \mid 1 \leq k \leq m,\ \ l_k \leq p_i,\ \ a_k \leq q_j \text{ e } v_k = V(i,j)\} \cup \{0\})$.
    $guilhotina(i,j) = nil$.
Para $i = 2$ até r
  Para $j = 2$ até s
    $n = max(k \mid 1 \leq k \leq r \text{ e } p_k \leq \lfloor \frac{p_i}{2} \rfloor)$.
    Para $x = 2$ até n
      $t = max(k \mid 1 \leq k \leq r \text{ e } p_k \leq p_i - p_x)$.
      Se $V(i,j) < V(x,j) + V(t,j)$.
        $V(i,j) = V(x,j) + V(t,j)$, $posicao(i,j) = p_x$ e $guilhotina(i,j) =$ 'V'.
    $n = max(k \mid 1 \leq k \leq s \text{ e } q_k \leq \lfloor \frac{q_j}{2} \rfloor)$.
    Para $y = 2$ até n
      $t = max(k \mid 1 \leq k \leq s \text{ e } q_k \leq q_j - q_y)$.
      Se $V(i,j) < V(i,y) + V(i,t)$
        $V(i,j) = V(i,y) + V(i,t)$, $posicao(i,j) = q_y$ e $guilhotina(i,j) =$ 'H'.

Observe que cada atribuição de valor à variável t pode ser feita em tempo $\mathcal{O}(\log r + \log s)$, simplesmente fazendo-se uma busca binária entre os pontos de discretização. Se utilizarmos o algoritmo DEE para calcular os pontos de discretização, o PCGV$_2$ pode ser implementado de forma a ter complexidade de tempo $\mathcal{O}(\delta_1 + \delta_2 + r^2 s \log r + r s^2 \log s)$, onde δ_1 e δ_2 representam a quantidade de combinações cônicas inteiras que produzem pontos de discretização da largura e da altura, respectivamente.

Para as instâncias do PCGV$_2$ onde $l_i > \frac{L}{k}$ e $a_i > \frac{A}{k}$ (k fixo e $i = 1, \ldots, m$), temos que δ_1, δ_2, r e s são polinomiais em m. Mais precisamente, δ_1 e δ_2 são menores ou iguais à combinação de m elementos tomados k a k. Além disso, $r \leq \delta_1$ e $s \leq \delta_2$. Dessa forma, para tais instâncias o PCGV$_2$PD requer tempo polinomial em m.

Podemos calcular um vetor X, com L posições, onde cada posição X_i contém $p(X_i)$. Analogamente, podemos ter um vetor Y, com A posições, onde cada posição Y_j contém $q(Y_j)$. Uma vez calculados os pontos de discretização, determinar os valores contidos nos vetores X e Y requer tempo $\mathcal{O}(L + A)$. Utilizando estes vetores, cada atribuição de valor à variável t pode ser feita em tempo constante. Neste caso, uma implementação do algoritmo PCGV$_2$, usando o DEE como sub-rotina, teria complexidade de tempo $\mathcal{O}(\delta_1 + \delta_2 + L + A + r^2 s + r s^2)$.

Calculando os pontos de discretização com o algoritmo DPD e utilizando os vetores X e Y descritos no parágrafo anterior, o PCGV$_2$PD pode ser implementado de forma a ter complexidade de tempo $\mathcal{O}(mL + mA + r^2 s + r s^2)$. Em qualquer caso, a quantidade de memória requerida pelo algoritmo PCGV$_2$PD é $\mathcal{O}(rs)$.

6.7 Resultados Computacionais

Implementamos o algoritmo PCGV$_2$PD utilizando o DPD como sub-rotina[6] e avaliamos seu desempenho resolvendo as instâncias do PCGV$_2$ disponíveis na OR-LIBRARY[7]. Tal biblioteca é uma coleção de instâncias de testes para uma grande variedade de problemas na área de pesquisa operacional. Uma descrição desta biblioteca e de seus objetivos é dada por Beasley [Bea90].

O PCGV$_2$PD foi implementado utilizando-se a linguagem C e o código executável gerado pelo compilador *gcc* versão 2.95.4 (*Debian prerelease*). Os testes foram executados num computador com dois processadores *AMD Athlon MP 1800+*, clock de 1.5 Ghz, 3.5 GB de memória principal e sistema operacional *Linux* (distribuição *Debian GNU/Linux 3.0*).

Durante a elaboração desta tese, estavam disponíveis na OR-LIBRARY 13 instâncias do PCGV$_2$, denominadas $gcut1, \ldots, gcut13$. Para todas estas instâncias, exceto para a instância $gcut13$, já haviam sido encontradas soluções ótimas [Bea85a]. Nestas instâncias as placas são quadradas, com dimensões variando entre 250 e 3000, e a quantidade de

[6] A menos que L ou A sejam muito grandes, digamos da ordem de milhões, é mais vantajoso, em termos práticos, calcular os pontos de discretização utilizando o algoritmo DPD em vez do algoritmo DEE.

[7] `http://mscmga.ms.ic.ac.uk/info.html`

itens varia entre 10 e 50 itens. O valor de cada item é igual à sua área. Ademais, em todas estas instâncias rotações ortogonais não são permitidas. No Apêndice A fornecemos a descrição completa destas instâncias.

Na Tabela 6.1 apresentamos detalhes das instâncias (quantidade de itens e dimensões da placa), a quantidade de pontos de discretização da largura (r) e da altura (s), o valor da solução encontrada pelo algoritmo PCGV$_2$PD (que é uma solução ótima), o percentual de desperdício e o tempo médio gasto para resolvê-las. Cada instância foi resolvida 100 vezes e os tempos apresentados foram obtidos calculando-se a média do tempo gasto nestas 100 resoluções.

Destacamos a resolução da instância $gcut13$ cuja solução ótima era desconhecida até então. Exibimos na Figura 6.2 a solução encontrada pelo algoritmo PCGV$_2$PD para esta instância.

Instância	Quantidade de Itens	Dimensões da Placa	r	s	Solução Ótima	Desperdício	Tempo (seg)
$gcut1$	10	(250, 250)	19	68	56460	9,664%	0,003
$gcut2$	20	(250, 250)	112	95	60536	3,142%	0,010
$gcut3$	30	(250, 250)	107	143	61036	2,342%	0,012
$gcut4$	50	(250, 250)	146	146	61698	1,283%	0,022
$gcut5$	10	(500, 500)	76	39	246000	1,600%	0,004
$gcut6$	20	(500, 500)	120	95	238998	4,401%	0,008
$gcut7$	30	(500, 500)	126	179	242567	2,973%	0,017
$gcut8$	50	(500, 500)	262	225	246633	1,347%	0,062
$gcut9$	10	(1000, 1000)	41	91	971100	2,890%	0,006
$gcut10$	20	(1000, 1000)	155	89	982025	1,798%	0,009
$gcut11$	30	(1000, 1000)	326	238	980096	1,990%	0,066
$gcut12$	50	(1000, 1000)	363	398	979986	2,001%	0,140
$gcut13$	32	(3000, 3000)	2425	1891	8997780	0,025%	144,915

Tabela 6.1: Soluções do PCGV$_2$PD para as instâncias $gcut1, \ldots, gcut13$.

Experimentamos também resolver as instâncias $gcut1, \ldots, gcut13$ permitindo rotações (chamamos tais instâncias de $gcut1r, \ldots, gcut13r$). Apresentamos na Tabela 6.2 os resultados obtidos nos testes computacionais realizados com estas instâncias. Os padrões cor-

respondentes às soluções ótimas encontradas para as instâncias $gcut1$, ..., $gcut13$, $gcut1r$, ..., $gcut13r$ podem ser visualizados em `http://www.ime.usp.br/~glauber/gcut`.

Instância	Quantidade de Itens	Dimensões da Placa	r	s	Solução Ótima	Desperdício	Tempo (seg)
$gcut1r$	10	(250, 250)	92	92	58136	6,982%	0,008
$gcut2r$	20	(250, 250)	142	142	60611	3,022%	0,021
$gcut3r$	30	(250, 250)	152	152	61626	1,398%	0,026
$gcut4r$	50	(250, 250)	166	166	62265	0,376%	0,042
$gcut5r$	10	(500, 500)	154	154	246000	1,600%	0,019
$gcut6r$	20	(500, 500)	201	201	240951	3,620%	0,032
$gcut7r$	30	(500, 500)	232	232	245866	1,654%	0,053
$gcut8r$	50	(500, 500)	292	292	247787	0,885%	0,135
$gcut9r$	10	(1000, 1000)	173	173	971100	2,890%	0,023
$gcut10r$	20	(1000, 1000)	294	294	982025	1,798%	0,071
$gcut11r$	30	(1000, 1000)	461	461	980096	1,990%	0,270
$gcut12r$	50	(1000, 1000)	363	363	988694	1,131%	0,325
$gcut13r$	32	(3000, 3000)	2469	2469	9000000	0,000%	280,247

Tabela 6.2: Soluções do $PCGV_2PD$ para as instâncias $gcut1r, \ldots, gcut13r$.

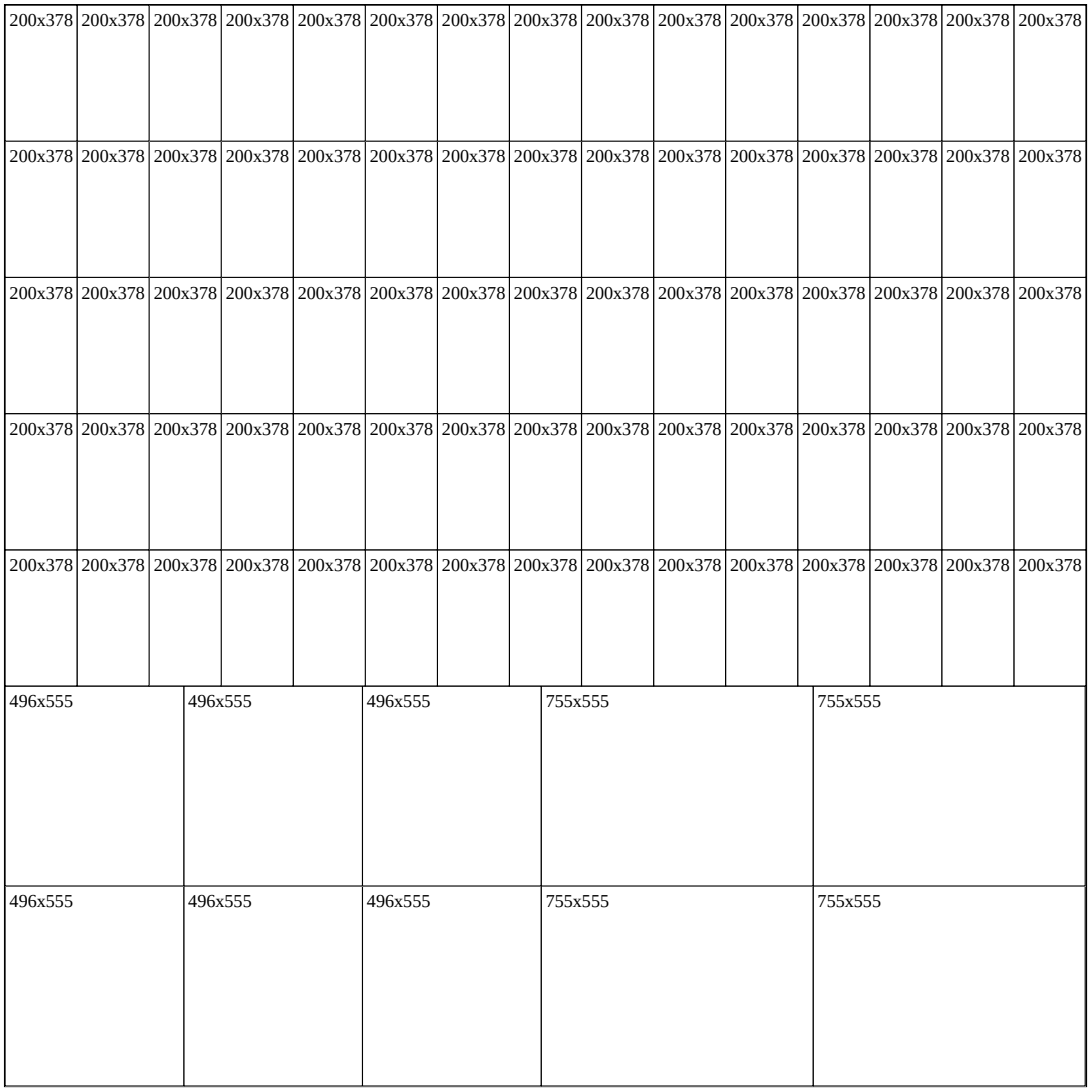


Figura 6.2: Solução ótima encontrada pelo algoritmo PCGV$_2$ para a instância *gcut13*

Capítulo 7

O Método de Geração de Colunas Aplicado ao $PCGD_2$

7.1 Introdução

Veremos neste capítulo como aplicar o método de geração de colunas ao problema de corte de guilhotina bidimensional com demandas ($PCGD_2$). Antes, porém, discutiremos a aplicação deste método ao problema de corte de estoque unidimensional com demandas ($PCED_1$), que pode ser visto como um caso particular do $PCGD_2$. Em seguida, mostraremos como adaptar o método de geração de colunas para o $PCGD_2$.

7.2 Geração de Colunas para o $PCED_1$

O método de geração de colunas foi primeiramente proposto para o $PCED_1$ por Gilmore e Gomory [GG61, GG63, GG66]. Este problema consiste em: dada uma quantidade ilimitada de objetos, genericamente denominados de *barras*, de comprimento L, e uma lista de m itens, cada item i com comprimento $l_i \leq L$ e demanda não-nula $d_i \in \mathbb{N}$ $(i = 1, \ldots, m)$, queremos determinar o menor número de barras necessárias para atender a demanda. Obviamente também estamos interessados em determinar como as barras devem ser cortadas.

Inicialmente precisamos formular o $PCED_1$ como um problema de programação linear inteira (PLI). Para fazer isso, representamos cada padrão j por um vetor p_j, cujo i-ésimo elemento indica o número de vezes que o item i ocorre nesse padrão. O problema agora consiste em considerar os padrões viáveis e decidir quantas vezes cada padrão deve ser

utilizado de modo a atender a demanda, minimizando o número total de barras utilizadas. Supondo existir n padrões viáveis, introduzimos um vetor x cujos elementos são inteiros x_j $(j = 1, \ldots, n)$, onde x_j indica quantas vezes o padrão j é selecionado. Assim, denotando por P a matriz $m \times n$ cujas colunas são os vetores $p_1, \ldots, p_n$, e representando por d o vetor das demandas, o problema pode ser assim formulado:

$$\begin{aligned} &\text{minimize} \quad \sum_{j=1}^{n} x_j \\ &Px = d \\ &x_j \geq 0 \quad \text{e inteiro} \qquad j = 1, \ldots, n. \end{aligned} \tag{7.1}$$

A formulação acima traz consigo duas dificuldades em termos computacionais. A primeira é determinar a matriz P (que pode ter um número exponencial de colunas); a segunda é resolver um problema de programação linear inteira (que em geral são $\mathcal{NP}$-difíceis). Para se desvencilhar destas dificuldades, Gilmore e Gomory propuseram o método de geração de colunas, que consiste em resolver a relaxação linear de (7.1), formulada abaixo, gerando gradativamente as colunas de P.

$$\begin{aligned} &\text{minimize} \quad \sum_{j=1}^{n} x_j \\ &Px = d \\ &x_j \geq 0 \qquad j = 1, \ldots, n. \end{aligned} \tag{7.2}$$

Podemos iniciar a resolução de (7.2) tomando a matriz identidade de ordem m, que chamaremos de B, como base da matriz P. Observe que cada coluna de B corresponde a um padrão viável pois todo item cabe na barra. Sem perda de generalidade, vamos supor que as colunas de B correspondem às primeiras m colunas de P. Denotaremos por x_B a parte do vetor x correspondente às colunas de B. Obviamente, para toda coluna j de P que não faz parte de B, $x_j = 0$. No início da primeira iteração, fazendo $x_B = d$ temos uma solução x para $Px = d$.

Em cada iteração determinamos um vetor y tal que $y^T B = \mathbb{1}^T$ [1]. Utilizando y, geramos uma nova coluna $z = (z_1, \ldots, z_m)$ resolvendo o seguinte PLI, que corresponde a uma instância do problema da mochila.

$$\begin{array}{l} \text{maximize} \quad y^T z \\ \sum_{i=1}^{m} z_i l_i \leq L \\ z_i \geq 0 \quad \text{e} \quad \text{inteiro} \qquad i = 1, \ldots, m. \end{array} \tag{7.3}$$

Após resolver (7.3), se $y^T z > 1$, resolvemos $Bw = z$ e determinamos:

$$t = min(\frac{x_j}{w_j} \mid 1 \leq j \leq m,\ w_j > 0), \tag{7.4}$$

e

$$s = min(j \mid 1 \leq j \leq m,\ \frac{x_j}{w_j} = t). \tag{7.5}$$

Em seguida substituímos os valores da coluna s de B pelos valores do vetor z (nova coluna), obtendo uma nova base B'. Utilizando t e a solução corrente x, calculamos uma nova solução viável x', como delineado na seguinte proposição.

Proposição 7.2.1 *Seja x uma solução viável de (7.2), t e s soluções de (7.4) e (7.5), respectivamente, $x'_j = x_j - w_j t$ para $j = 1, \ldots, s-1, s+1, \ldots, m$, $x'_s = t$ e $x'_j = 0$ para $j = m+1, \ldots, n$. Então x' é solução viável de (7.2).*

Prova. Primeiramente vamos mostrar que $x'_j \geq 0$ $(j = 1, \ldots, n)$. Observe que $t \geq 0$, pois $x_s \geq 0$ e $w_s > 0$, portanto $x'_s \geq 0$. Resta mostrar que $x'_j \geq 0$ para $j = 1, \ldots, s-1, s+$

[1] $\mathbb{1}$ é um vetor cujos elementos são todos iguais a 1 (a quantidade de elementos de $\mathbb{1}$ é determinada pelo contexto em que ele aparece).

$1, \ldots, m$. Note que $t = \frac{x_s}{w_s}$ e $\frac{x_j}{w_j} \geq \frac{x_s}{w_s}$, logo $x_j w_s - x_s w_j \geq 0$ $(j = 1, \ldots, m)$. Dessa forma, temos que

$$x'_j = x_j - w_j t = x_j - w_j \frac{x_s}{w_s} = \frac{x_j w_s - x_s w_j}{w_s} \geq 0.$$

Vamos mostrar agora que $Px' = d$. Seja $x_B = (x_1, \ldots, x_m)^T$ e $x'_B = (x'_1, \ldots, x'_m)^T$. Observe que $Px' = B'x'_B$, $Bx_B = d$ e que B difere de B' apenas na coluna s, sendo z a coluna s de B'. Ademais, $x_s - w_s t = x_s - w_s \frac{x_s}{w_s} = 0$. Assim,

$$Px' = B'x'_B = B(x_B - wt) + zt = Bx_B - Bwt + zt = d - zt + zt = d.$$

■

O valor da função objetivo de (7.2) no ponto x' não é pior que o valor da função objetivo de (7.2) no ponto x, conforme demonstramos a seguir.

Proposição 7.2.2 *Para x e x' mencionados na Proposição 7.2.1, temos que*

$$\sum_{j=1}^{n} x'_j \leq \sum_{j=1}^{n} x_j.$$

Prova. Sejam $x_B = (x_1, \ldots, x_m)^T$ e $x'_B = (x'_1, \ldots, x'_m)^T$. Lembrando que $x_s - w_s t = 0$, $y^T B = \mathbb{1}^T$, $Bw = z$, $t \geq 0$ e $y^T z > 1$, temos que

$$\sum_{j=1}^{n} x'_j = \mathbb{1}^T x'_B = \mathbb{1}^T (x_B - wt) + x'_s = \mathbb{1}^T x_B - \mathbb{1}^T wt + t = \sum_{j=1}^{n} x_j - y^T Bwt + t$$

$$= \sum_{j=1}^{n} x_j - y^T zt + t = \sum_{j=1}^{n} x_j + t(1 - y^T z) \leq \sum_{j=1}^{n} x_j.$$

■

Começamos então uma nova iteração com $x = x'$ e $B = B'$. Se $y^T z \leq 1$, então a solução x corrente é ótima. Para demonstrar este fato podemos usar o dual de (7.2), formulado a seguir.

$$\begin{aligned} &\text{minimize } \ y^T d \\ &y^T P \leq \mathbb{1}^T. \end{aligned} \tag{7.6}$$

O *Teorema das Folgas Complementares*, enunciado abaixo de uma forma conveniente aos nossos propósitos, estabelece relações entre soluções ótimas de (7.2) e (7.6).

Teorema 7.2.3 (Folgas Complementares) *Dada uma solução viável x de (7.2) e uma solução viável y de (7.6), x e y são soluções ótimas de (7.2) e (7.6), respectivamente, se e somente se*

$$x_j(1 - \sum_{i=1}^{m} y_i P_{i,j}) = 0 \ \text{ para } \ j = 1, \ldots, n \quad \text{e} \quad (d_i - \sum_{j=1}^{n} P_{i,j} x_j) y_i = 0 \ \text{ para } \ i = 1, \ldots, m.$$

Uma demonstração deste teorema pode ser obtida em [Chv80], pelo que deixamos de transcrevê-la aqui. Agora estamos em condições de provar que se a solução de (7.3) tiver valor menor ou igual a 1 então x é solução ótima de (7.2).

Proposição 7.2.4 *Dada uma solução viável x de (7.2) e um vetor y tal que $y^T B = \mathbb{1}^T$, se a solução z de (7.3) for tal que $y^T z \leq 1$ então x é solução ótima de (7.2).*

Prova. Observe que $y^T P_j \leq 1$ $(j = 1, \ldots, n)$, caso contrário tal P_j seria uma solução de (7.3) com valor maior que 1. Portanto y é solução viável de (7.6). Note que para todo j tal que $x_j > 0$, P_j é uma coluna de B; logo $\sum_{i=1}^{m} y_i P_{i,j} = 1$, pois $y^T B = \mathbb{1}^T$. Assim, $x_j(1 - \sum_{i=1}^{m} y_i P_{i,j}) = 0$ $(j = 1, \ldots, n)$. Ademais, $\sum_{j=1}^{n} P_{i,j} x_j = d_i$ $(i = 1, \ldots, m)$, pois x é solução viável de (7.2). Portanto, pelo *Teorema das Folgas Complementares*, x é solução ótima de (7.2). ■

Temos então o algoritmo que chamaremos de *Simplex com Geração de Colunas para o PCE$_1$*, ou simplesmente SimplexGC$_1$, cujos detalhes são apresentados a seguir.

Algoritmo 7.1 SimplexGC$_1$

Entrada: (L, l, d) onde $l = (l_1, \ldots, l_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução ótima de (7.2): minimize $\sum_{j=1}^{n} x_j$

$$Px = d$$
$$x_j \geq 0 \qquad j = 1, \ldots, n.$$

(onde P é a matriz dos padrões viáveis)

1 Faça $x = d$ e seja B a matriz identidade de ordem m.

2 Resolva $y^T B = \mathbb{1}^T$.

3 Gere uma nova coluna resolvendo (7.3):

$$\text{maximize } y^T z$$
$$z^T l \leq L$$
$$z_i \geq 0 \quad \text{e inteiro} \qquad i = 1, \ldots, m\ .$$

4 Se $y^T z \leq 1$, devolva B e x e pare (tal x corresponde apenas às colunas de B).

5 Caso contrário, resolva $Bw = z$.

6 Calcule $t = min(\frac{x_j}{w_j} \mid 1 \leq j \leq m,\ w_j > 0)$.

7 Calcule $s = min(j \mid 1 \leq j \leq m,\ \frac{x_j}{w_j} = t)$.

8 Para $i = 1$ até m faça

8.1 $B_{i,s} = z_i$.

8.2 Se $i = s$ então $x_i = t$; caso contrário, $x_i = x_i - w_i t$.

9 Retorne ao passo 2.

Ainda não deixamos claro como executar o passo 3 do algoritmo SimplexGC$_1$. Observe, no entanto, que toda solução ótima de (7.3) tem que satisfazer

$$L - \sum_{i=1}^{m} l_i z_i < l_m, \tag{7.7}$$

pois se isso não ocorresse seria possível incrementar z_m de uma unidade. Chamamos as soluções viáveis de (7.3) que satisfazem (7.7) de *soluções sensatas*.

Podemos executar o passo 3 do SimplexGC$_1$ utilizando o algoritmo MOCHILA, que consiste basicamente em enumerar as soluções sensatas, começando por aquelas que pare-

[1] Se P é uma matriz, então $P_{i,j}$ é o elemento que está na linha i e na coluna j de P. Ademais, P_j denota a coluna j de P, e P^i é a linha i de P.

cem ser mais promissoras, buscando entre elas uma solução ótima. Apresentamos a seguir os detalhes do algoritmo.

Algoritmo 7.2 Mochila

Entrada: $(L, l_1, \ldots, l_m, y_1, \ldots, y_m)$.

Saída: $z_1, \ldots, z_m \in \mathbb{N}$ tais que $\sum_{i=1}^{m} l_i z_i \leq L$ e $\sum_{i=1}^{m} y_i z_i$ é máximo.

1 Faça $M = 0$ e $k = 0$.

2 Ordene os itens em ordem decrescente de $\frac{y_i}{l_i}$ (custo relativo).

3 Para $j = k + 1$ até m faça

3.1 $z_j = \lfloor (L - \sum_{i=1}^{j-1} l_i z_i)/l_j \rfloor$.

4 Se $M \leq \sum_{i=1}^{m} y_i z_i$ faça $z^* = z$ e $M = \sum_{i=1}^{m} y_i z_i^*$.

5 Procure $k = max(\{i \mid 1 \leq i < m \text{ e } z_i > 0,\ \sum_{j=1}^{i-1} y_j z_j + y_i(z_i - 1) + \frac{y_{i+1}}{l_{i+1}}(L - \sum_{j=1}^{i-1} l_j z_j - l_i(z_i - 1)) > M\} \cup \{0\})$.

6 Se $k = 0$ então devolva z^* e pare.

7 Faça $z_k = z_k - 1$ e retorne ao passo 3.

Ao enumerar as soluções sensatas, podemos evitar que soluções claramente inferiores à melhor solução já encontrada sejam analisadas. Tal procedimento limita sobremaneira o espaço de busca, tornando o algoritmo sensivelmente mais rápido.

Observe que no passo 2 colocamos os itens em ordem decrescente de custo relativo. No passo 5, ao procurar um valor para k, consideramos possíveis soluções $\bar{z}$ tais que $\bar{z}_i = z_i$ $(i = 1, \ldots, k-1)$ e $\bar{z}_k = z_k - 1$. A idéia é determinar a priori, quaisquer que sejam os valores de $\bar{z}_{k+1}, \ldots, \bar{z}_m$, se $\bar{z}$ tem chance de ser melhor que z^*. Cada item $k+1, \ldots, m$ tem custo relativo menor ou igual a $\frac{y_{k+1}}{l_{k+1}}$, portanto

$$\sum_{i=k+1}^{m} y_i \bar{z}_i \leq \frac{y_{k+1}}{l_{k+1}}(L - \sum_{i=1}^{k-1} l_i z_i - l_k(z_k - 1)).$$

Dessa forma, para todas as soluções $\bar{z}$ com os valores das primeiras k posições fixadas como acabamos de descrever, vale que

$$\sum_{i=1}^{m} y_i \bar{z}_i \leq \sum_{i=1}^{k-1} y_i z_i + y_k(z_k - 1) + \frac{y_{k+1}}{l_{k+1}}(L - \sum_{i=1}^{k-1} l_i z_i - l_k(z_k - 1)). \tag{7.8}$$

Como desejamos que o valor de uma solução $\bar{z}$ seja maior que M, podemos impor que a seguinte desigualdade seja satisfeita ao escolher o valor de k, caso contrário $\bar{z}$ não tem chance de ser melhor que z^*.

$$\sum_{j=1}^{k-1} y_j z_j + y_i(z_k - 1) \; + \; \frac{y_{k+1}}{l_{k+1}}(L - \sum_{j=1}^{k-1} l_j z_j - l_k(z_k - 1)) > M. \tag{7.9}$$

Podemos dizer que o algoritmo MOCHILA executa uma busca em profundidade numa árvore, onde cada ramo da árvore (um caminho desde a raiz até uma folha) representa uma solução sensata. A desigualdade (7.9) permite podar esta árvore fazendo com que o algoritmo encontre uma solução mais rapidamente. Trata-se portanto de um algoritmo de enumeração do tipo *branch-and-bound*. Apesar de ser exponencial no pior caso, este algoritmo se mostra bastante satisfatório na prática.

Voltando ao algoritmo SimplexGC$_1$, a condição suficiente para que a nova coluna possa entrar na base é $y^T z > 1$. Dessa forma, não precisamos encontrar uma solução ótima de (7.3). Implementamos uma versão do SimplexGC$_1$ que utiliza o algoritmo MOCHILA aqui descrito e outra versão em que o algoritmo MOCHILA pára ao encontrar uma solução com valor maior que 1 (não necessariamente ótima). Os resultados dos testes que realizamos (veja [Cin99]) indicam que buscar soluções ótimas de (7.3) faz com que seja necessário gerar um número menor de colunas, compensando assim o maior esforço na geração de colunas.

Com relação à quantidade de colunas geradas pelo SimplexGC$_1$, Klee e Minty [KM72] exibiram uma família de problemas de programação linear, criada artificialmente a partir de deformações do cubo n-dimensional, para a qual o algoritmo Simplex, com regra de pivotação de Dantzig-Wolfe, executa um número exponencial de iterações. Outros autores provaram que o Simplex pode requerer tempo exponencial mesmo utilizando outras regras de pivotação [Zad73].

No entanto, os testes computacionais que realizamos indicam que o número médio de colunas geradas pelo SimplexGC$_1$ é $\mathcal{O}(m^2)$. Isto parece estar de acordo com resultados

teóricos obtidos por diversos pesquisadores a respeito do tempo requerido pelo Simplex no caso médio [AMT84, Bor90].

Infelizmente a solução ótima de (7.2), obtida pelo SimplexGC$_1$, não é necessariamente inteira. De fato, o valor de uma solução ótima de (7.2) pode não ser igual ao valor de uma solução ótima de (7.1). Para contornar este problema, Gilmore e Gomory propuseram, após achar uma solução ótima de (7.2), arredondar para cima o valor das variáveis (isto é, arredondar cada x_i para $\lceil x_i \rceil$), obtendo assim uma solução inteira. No entanto, este procedimento pode acarretar a produção de itens em quantidade superior à demanda, e conduz a uma solução inteira cujo valor pode eventualmente estar longe do valor de uma solução inteira ótima.

Diversos métodos heurísticos têm sido propostos para se obter soluções inteiras com valor mais próximo do ótimo [WG96, Cin99]. Destacamos o algoritmo por nós proposto em [Cin99], chamado de HÍBRIDO. Demonstramos que o valor da solução encontrada pelo algoritmo HÍBRIDO difere do valor de uma solução inteira ótima de no máximo 1, se verdadeira uma conjectura proposta por Scheithauer e Terno [ST95]. Tal conjectura estabelece a seguinte relação entre o valor de uma solução ótima de (7.2) e o valor de uma solução ótima de (7.1).

Conjectura 7.2.5 (Scheithauer & Terno' 1995) *: Sejam x e x^* soluções ótimas de (7.2) e (7.1), respectivamente. Então*

$$\sum_{i=1}^{m} x_i \leq \left\lceil \sum_{i=1}^{m} x_i^* \right\rceil + 1.$$

Em experimentos computacionais realizados por diversos autores, onde foram determinadas soluções inteiras ótimas, verificou-se que em todas as instâncias testadas a diferença entre os valores de x e x^* era menor que 2 [ST92, WG93, ST95, Cin98]. De fato, até a data em que esta tese foi finalizada, a maior diferença conhecida, entre os valores de x e x^* é de 1,1666... [RST02]. Ademais, a validade desta conjectura já foi estabelecida para diversas classes de instâncias do PCED$_1$ [ST95, ST97, NST98].

Dizemos que o algoritmo HÍBRIDO é um algoritmo *quase-exato*, se verdadeira a conjectura de Scheithauer e Terno, visto que ele sempre encontra soluções ótimas ou *quase-ótimas*.

Na próxima seção, veremos que é possível adaptar o algoritmo SimplexGC$_1$ para o PCGD$_2$, efetuando uma modificação na maneira de gerar novas colunas.

7.3 Adaptando o Método de Geração de Colunas para o PCGD$_2$

No caso unidimensional, gerar uma nova coluna equivale a encontrar uma solução de (7.3) com valor maior que 1. Uma solução de (7.3) indica uma coleção de itens com a seguinte propriedade: a soma dos comprimentos dos itens da coleção não excede o comprimento da barra. Esta coleção é composta por z_i itens de comprimento l_i ($i = 1, \ldots, m$). Esta propriedade garante a existência de um padrão contendo a coleção de itens.

Já no caso bidimensional, a propriedade citada no parágrafo anterior não garante a existência de um padrão contendo a coleção de itens expressa pela solução de (7.3). Mesmo que alterássemos a desigualdade $\sum_{i=1}^{m} z_i l_i \leq L$ para $\sum_{i=1}^{m} z_i l_i a_i \leq LA$, ou seja, exigíssemos que a soma das áreas dos itens da coleção não excedesse a área da placa, ainda assim, tal propriedade não garantiria a existência de um padrão viável. Existe a necessidade de impedir a sobreposição dos itens dentro do padrão.

No entanto, o algoritmo PCGV$_2$PD, explicado na Seção 6.6, é capaz de gerar padrões guilhotináveis. Além disso, se cada item i tem valor y_i e aparece z_i vezes num padrão produzido pelo PCGV$_2$PD ($i = 1, \ldots, m$), então $\sum_{i=1}^{m} y_i z_i$ é máximo. Isto é exatamente o que precisamos para gerar novas colunas. Dessa forma, tudo o que precisamos fazer para adaptar o algoritmo SimplexGC$_1$ para o PCGD$_2$ é utilizar o algoritmo PCGV$_2$PD para gerar novas colunas. A seguir, detalhamos o algoritmo SimplexGC$_2$ que resolve a formulação (7.2) correspondente a uma instância do PCGD$_2$.

Ao executar o passo 3 do SimplexGC$_2$, as matrizes *guilhotina* e *posicao*, calculadas pelo algoritmo PCGV$_2$PD, nos permitem reconstruir o padrão representado por z. Fica implícito na descrição do algoritmo que esta informação deve ser devolvida, pois também estamos interessados em saber como cortar as placas.

Como já mencionamos anteriormente, a formulação (7.2) pode não ter solução ótima onde todas as variáveis sejam inteiras. Na próxima seção discutimos como lidar com esta dificuldade.

Algoritmo 7.3 SimplexGC$_2$

Entrada: (L, A, l, a, d) onde $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução ótima de (7.2): minimize $\sum_{j=1}^{n} x_j$

$$Px = d$$
$$x_j \geq 0 \qquad j = 1, \ldots, n.$$

(onde P é a matriz dos padrões viáveis)

1 Faça $x = d$ e seja B a matriz identidade de ordem m.
2 Resolva $y^T B = \mathbb{1}^T$.
3 Gere uma nova coluna z executando o algoritmo PCGV$_2$PD com parâmetros L, A, l, a, y.
4 Se $y^T z \leq 1$, devolva B e x e pare (tal x corresponde apenas às colunas de B).
5 Caso contrário, resolva $Bw = z$.
6 Calcule $t = min(\frac{x_j}{w_j} \mid 1 \leq j \leq m,\ w_j > 0)$.
7 Calcule $s = min(j \mid 1 \leq j \leq m,\ \frac{x_j}{w_j} = t)$.
8 Para $i = 1$ até m faça
  8.1 $B_{i,s} = z_i$.
  8.2 Se $i = s$ então $x_i = t$; caso contrário, $x_i = x_i - w_i t$.
9 Retorne ao passo 2.

7.4 O Algoritmo PCGD$_2$GC

Explicaremos agora como podemos obter uma solução inteira a partir das soluções encontradas pelo SimplexGC$_2$. O processo é iterativo. Cada iteração inicia com uma instância I do PCGD$_2$ e consiste basicamente em resolver (7.2) com o SimplexGC$_2$ obtendo B e x. Se x for inteira, devolvemos B e x e paramos. Caso contrário, calculamos $x^* = (x_1^*, \ldots, x_m^*)$ fazendo $x_i^* = \lfloor x_i \rfloor$ $(i = 1, \ldots, m)$. Adotando esta nova solução, uma parte da demanda dos itens não será atendida. Mais precisamente, a demanda não atendida de cada item i será $d_i^* = d_i - \sum_{j=1}^{m} B_{i,j} x_j^*$.

Fazendo $d^* = (d_1^*, \ldots, d_m^*)$, temos então uma instância residual $I^* = (L, A, l, c, d^*)$ (podemos ter o cuidado de eliminar de I^* os itens que porventura tenham demanda nula). Se algum $x_i^* > 0$ $(i = 1, \ldots, m)$, uma parte da demanda é atendida pela solução x^*. Neste caso devolvemos B e x, fazemos $I = I^*$ e começamos uma nova iteração. Se $x_i^* = 0$ $(i = 1, \ldots, m)$, nenhuma parte da demanda é atendida por x^*. Resolvemos então a instância I^* com o algoritmo HFF, explicado na Seção 4.4.

Apresentamos a seguir o algoritmo PCGD$_2$GC que implementa o processo iterativo que acabamos de descrever.

Algoritmo 7.4 PCGD$_2$GC

Entrada: Uma instância $I = (L, A, l, a, d)$ do PCGD$_2$, onde $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I

1 Execute o algoritmo SimplexGC$_2$ com parâmetros L, A, l, a, d obtendo B e x.
2 Para $i = 1$ até m faça $x_i^* = \lfloor x_i \rfloor$.
3 Se $x_i^* > 0$ para algum i, $1 \leq i \leq m$, então
  3.1 Devolva B e $x_1^*, \ldots, x_m^*$ (mas não pare).
  3.2 Para $i = 1$ até m faça
    3.2.1 Para $j = 1$ até m faça $d_i = d_i - B_{i,j} x_j^*$.
  3.3 Faça $m' = 0$.
  3.4 Para $i = 1$ até m faça
    3.4.1 Se $d_i > 0$ faça $m' = m' + 1$, $l_{m'} = l_i$, $a_{m'} = a_i$ e $d_{m'} = d_i$.
  3.5 Se $m' = 0$ então pare.
  3.6 Faça $m = m'$ e volte ao passo 1.
4 $l' = \emptyset$, $a' = \emptyset$. `/* l' e a' são multiconjuntos */`
5 Para $i = 1$ até m faça
  5.1 Para $j = 1$ até d_i faça
    5.1.1 $l' = l' \cup \{l_i\}$, $a' = a' \cup \{a_i\}$.
6 Devolva a solução do algoritmo HFF executado com parâmetros L, A, l', a'.

Observe que em cada iteração ou uma parte da demanda é atendida ou passamos para o passo 4. Isto garante que após um número finito de iterações, toda a demanda terá sido atendida (uma parte dela eventualmente no passo 4). Na realidade, sobre o número de iterações do algoritmo, vale o seguinte resultado.

Proposição 7.4.1 *O passo 3.6 do algoritmo PCGD$_2$GC é executado no máximo m vezes.*

Prova. Ao executar o passo 3.6, seja d^* o vetor-demanda da instância residual. Note que $d^* = Ax - Ax^*$. Faça $x' = x - x^*$. Observe que tal x' é uma solução ótima da

formulação (7.2) correspondente à instância residual pois $Ax' = Ax - Ax^* = d^*$ e $x'_i = x_i - x^*_i = x_i - \lfloor x_i \rfloor \geq 0$ $(i = 1, \ldots, m)$. Dessa forma, na iteração seguinte, o valor da solução fornecida pelo SimplexGC$_2$ será $\sum_{i=1}^{m} x'$.

Se uma nova iteração é iniciada, então $\sum_{i=1}^{m} x^* \geq 1$. Além disso, $\sum_{i=1}^{m} x' = \sum_{i=1}^{m} x - \sum_{i=1}^{m} x^*$. Dessa forma, temos que $\sum_{i=1}^{m} x' \leq \sum_{i=1}^{m} x - 1$. Isto significa que, ao final de cada iteração, o valor ótimo de uma solução da instância residual decresce de pelo menos 1. Como no final da primeira iteração $\sum_{i=1}^{m} x' < m$, após no máximo m iterações teremos $\sum_{i=1}^{m} x' < 1$. Sendo assim, na iteração seguinte $\sum_{i=1}^{m} x < 1$ e consequentemente $\sum_{i=1}^{m} x^* = 0$. Dessa forma, o algoritmo não mais executará o passo 3.6, prosseguindo então para o passo 4. ■

Vimos assim que o número de chamadas ao SimplexGC$_2$ é polinomial. Ademais, o número de colunas geradas pelo SimplexGC$_2$ também é polinomial, no caso médio, conforme discutimos na Seção 7.2. Observe no entanto que o cálculo de l' e a' no passo 5 do PCGD$_2$GC pode levar tempo exponencial. Este passo é necessário para transformar a representação da última instância residual numa entrada para o HFF, utilizado no passo seguinte. Além disso, o HFF também pode requerer tempo exponencial para resolver esta última instância.

Pode-se perguntar por que não utilizamos o algoritmo PCGD$_2$SH, explicado na Seção 5.3, que é polinomial. Temos duas razões para esta escolha: primeiro, gerar uma nova coluna com o algoritmo PCGV$_2$PD requer tempo que pode ser exponencial em m. Somente este fato já torna o PCGV$_2$GC exponencial, mesmo no caso médio. Além disso, o HFF tem razão de aproximação assintótica 2,125 enquanto que a razão de aproximação absoluta do PCGD$_2$SH é 4. Dessa forma, a utilização do HFF tende a produzir soluções de melhor qualidade.

Por outro lado, é verdade que se os itens não forem muito pequenos em relação às placas[2], o PCGV$_2$PD pode ser implementado de forma a requerer tempo polinomial (veja Seção 6.6). Neste caso, poderia ser adequado eliminar os passos 4 e 5 do PCGD$_2$GC e utilizar o PCGD$_2$SH em vez do HFF para resolver o último problema residual. Uma eventual diminuição na qualidade das soluções encontradas seria compensada, pelo menos em termos teóricos, pela garantia de que o tempo requerido pelo algoritmo PCGD$_2$GC seria polinomial em m, no caso médio.

Convém observar que o PCGD$_2$GC é capaz de resolver a variante do PCGD$_2$ em que rotações são permitidas, que chamaremos de PCGD$_2^r$. Para isto, basta que na chamada

[2]Mais precisamente, para k fixo, $l_i > \frac{L}{k}$ e $a_i > \frac{A}{k}$ $(i = 1, \ldots, m)$.

do algoritmo PCGV$_2$PD, feita no passo 3 do algoritmo SimplexGC$_2$, façamos a transformação explicada na Subseção 3.3.5. Chamaremos o SimplexGC$_2$ com esta alteração de SimplexGC$_2^r$. Com isto, poderemos produzir padrões com itens que tenham sofrido rotação. No entanto o algoritmo HFF, utilizado no passo 6 do PCGD$_2$GC, não tira proveito da possibilidade de empacotar os itens após rotações ortogonais. Utilizamos então o algoritmo FFDHR, descrito na Seção 5.5.

O algoritmo PCGD$_2^r$GC, descrito a seguir, é uma versão especializada do PCGD$_2$GC para o PCGD$_2^r$.

Algoritmo 7.5 PCGD$_2^r$GC

Entrada: Uma instância $I = (L, A, l, a, d)$ do PCGD$_2^r$ onde $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I

1 Execute o algoritmo SimplexGC$_2^r$ com parâmetros L, A, l, a, d obtendo B e x.
2 Para $i = 1$ até m faça $x_i^* = \lfloor x_i \rfloor$.
3 Se $x_i^* > 0$ para algum i, $1 \leq i \leq m$, então
  3.1 Devolva B e $x_1^*, \ldots, x_m^*$ (mas não pare).
  3.2 Para $i = 1$ até m faça
    3.2.1 Para $j = 1$ até m faça $d_i = d_i - B_{i,j} x_j^*$.
  3.3 Faça $m' = 0$.
  3.4 Para $i = 1$ até m faça
    3.4.1 Se $d_i > 0$ faça $m' = m' + 1$, $l_{m'} = l_i$, $a_{m'} = a_i$ e $d_{m'} = d_i$.
  3.5 Se $m' = 0$ então pare.
  3.6 Faça $m = m'$ e volte ao passo 1.
4 $l' = \emptyset$, $a' = \emptyset$.
5 Para $i = 1$ até m faça
  5.1 Para $j = 1$ até d_i faça
    5.1.1 $l' = l' \cup \{l_i\}$, $a' = a' \cup \{a_i\}$.
6 Devolva a solução do algoritmo FFDHR executado com parâmetros L, A, l', a'.

Observe que, assim como o algoritmo HFF, o FFDHR possui complexidade de tempo polinomial no tamanho de sua entrada, mas pode requerer tempo exponencial em $\mathcal{O}(m)$, que é o tamanho da entrada do PCGD$_2^r$GC.

7.5 Perturbando as Instâncias Residuais

Experimentamos uma outra modificação no algoritmo PCGD$_2$GC (e no PCGD$_2^r$GC) com o intuito de obter soluções de melhor qualidade. Tal modificação consiste basicamente no seguinte: se ao resolver uma instância, a solução devolvida pelo SimplexGC$_2$, após ser arredondada para baixo, for igual a zero, em vez de submeter esta instância para o algoritmo HFF (ou para o FFDHR) resolver, utilizamos o HFF (ou o FFDHR) para obter um *bom* padrão, atualizamos as demandas e, se houver demanda não atendida, retornamos ao passo 1.

Em nossa implementação, consideramos como *bom* um padrão produzido pelo HFF (ou pelo FFDHR) que apresente o menor desperdício de área. Eventualmente, outros critérios podem ser utilizados.

Chamaremos o algoritmo obtido com esta modificação de PCGD$_2$GC^p. Observe que a idéia básica deste algoritmo é *perturbar* as instâncias residuais cuja solução de sua relaxação linear, arredondada para baixo, seja igual a zero. Com este procedimento, espera-se que a solução obtida pelo SimplexGC$_2$ para a instância residual possua mais variáveis com valores maiores que 1.

Convém observar que com esta modificação, a garantia de que precisaremos executar no máximo $m+1$ chamadas ao SimplexGC$_2$, implícita na Proposição 7.4.1, não é mais válida. No entanto, é fácil perceber que o PCGD$_2$GC^p converge, pois cada vez que executamos o passo 1 do algoritmo, a demanda é estritamente menor do que a da iteração anterior. Após um número finito de iterações, a demanda terá sido totalmente atendida e então o algoritmo pára (no passo 3.5 ou no passo 7). Pode ocorrer, no entanto, que a convergência do algoritmo PCGD$_2$GC^p seja mais lenta que a do PCGD$_2$GC.

Podemos adaptar o algoritmo PCGD$_2$GC^p para o caso com rotações simplesmente trocando as chamadas aos algoritmos SimplexGC$_2$ e HFF por chamadas aos algoritmos SimplexGC$_2^r$ e FFDHR, respectivamente. Chamaremos o algoritmo obtido através desta adaptação de PCGD$_2^r$GC^p, no entanto não apresentaremos sua descrição detalhada, por ser ele muito semelhante ao PCGD$_2$GC^p, cujos detalhes são fornecidos a seguir.

Apresentamos na próxima seção os resultados computacionais obtidos ao aplicar os algoritmos PCGD$_2$GC, PCGD$_2$GC^p, PCGD$_2^r$GC e PCGD$_2^r$GC^p a diversas instâncias.

Algoritmo 7.6 PCGD$_2$GC^p

Entrada: Uma instância $I = (L, A, l, a, d)$ do PCGD$_2$, onde $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I

1 Execute o algoritmo SimplexGC$_2$ com parâmetros L, A, l, a, d obtendo B e x.
2 Para $i = 1$ até m faça $x_i^* = \lfloor x_i \rfloor$.
3 Se $x_i^* > 0$ para algum i, $1 \leq i \leq m$, então
  3.1 Devolva B e $x_1^*, \ldots, x_m^*$ (mas não pare).
  3.2 Para $i = 1$ até m faça
    3.2.1 Para $j = 1$ até m faça $d_i = d_i - B_{i,j} x_j^*$.
  3.3 Faça $m' = 0$.
  3.4 Para $i = 1$ até m faça
    3.4.1 Se $d_i > 0$ faça $m' = m' + 1$, $l_{m'} = l_i$, $a_{m'} = a_i$ e $d_{m'} = d_i$.
  3.5 Se $m' = 0$ então pare.
  3.6 Faça $m = m'$ e volte ao passo 1.
4 $l' = \emptyset$, $a' = \emptyset$.
5 Para $i = 1$ até m faça
  5.1 Para $j = 1$ até d_i faça
    5.1.1 $l' = l' \cup \{l_i\}$, $a' = a' \cup \{a_i\}$.
6 Devolva um padrão gerado pelo algoritmo HFF, executado com parâmetros L, A, l', a', que apresente o menor desperdício de área e atualize as demandas.
7 Se houver demanda não atendida, volte ao passo 1.

7.6 Resultados Computacionais

Os algoritmos foram implementados utilizando-se a linguagem C e o código executável gerado pelo compilador *gcc* versão 2.95.4 (*Debian prerelease*). Os testes executados num computador com dois processadores *AMD Athlon MP 1800+*, clock de 1.5 Ghz, 3.5 GB de memória principal e sistema operacional *Linux* (distribuição *Debian GNU/Linux 3.0*). Fizemos uso do software *Xpress-MP* [Xpr02] para resolver os sistemas de equações lineares que aparecem nos passos 2 e 5 do algoritmo SimplexGC$_2$.

7.6.1 Resolvendo Instâncias do $PCGD_2$

Não encontramos instâncias do $PCGD_2$ na OR-LIBRARY. Decidimos então usar as instâncias $gcut1, \ldots, gcut12$, atribuindo a cada item uma demanda gerada aleatoriamente entre 1 e 100. Chamamos tais instâncias de $gcut1d, \ldots, gcut12d$. As demandas foram geradas utilizando-se a função *rand* da linguagem PERL (versão 5.6.1) [WCO00] com a semente sendo inicializada com o número $(L + A) * m + m$, onde L e A são a largura e a altura da placa, respectivamente, e m é a quantidade de itens. Por exemplo, para a instância $gcut1d$ a semente da função *rand* foi o número 5010. As demandas utilizadas nestas instâncias aparecem no Apêndice A.

Utilizamos o algoritmo $PCGD_2GC$ para resolver as instâncias $gcut1d, \ldots, gcut12d$. Apresentamos na Tabela 7.1 o valor da solução encontrada pelo $PCGD_2GC$, o limite inferior (LI) fornecido pela solução de (7.2) para o valor de uma solução inteira ótima, a diferença percentual entre a solução do $PCGD_2GC$ e o valor do LI, o tempo gasto, a quantidade de colunas geradas, o valor da solução encontrada pelo HFF e o ganho percentual da solução do $PCGD_2GC$ em relação à solução do HFF. Cada instância foi resolvida 10 vezes e os tempos apresentados foram obtidos calculando-se a média do tempo gasto nestas 10 resoluções.

Instância	Solução do $PCGD_2GC$	Limite Inferior (LI)	Diferença em relação ao LI	Tempo (seg)	Colunas Geradas	Solução do HFF	Ganho em relação ao HFF
$gcut1d$	294	294	0,000%	0,059	9	295	0,339%
$gcut2d$	345	345	0,000%	0,585	68	402	14,179%
$gcut3d$	333	332	0,301%	2,340	274	393	14,834%
$gcut4d$	838	836	0,239%	11,693	820	977	11,323%
$gcut5d$	198	197	0,507%	0,088	18	198	0,000%
$gcut6d$	344	343	0,291%	0,362	101	418	17,308%
$gcut7d$	591	591	0,000%	1,184	136	615	4,523%
$gcut8d$	691	690	0,145%	30,361	952	764	9,555%
$gcut9d$	131	131	0,000%	0,068	11	143	7,092%
$gcut10d$	293	293	0,000%	0,172	20	335	12,537%
$gcut11d$	331	330	0,303%	8,570	222	353	6,232%
$gcut12d$	673	672	0,149%	39,032	485	727	7,428%
Média			**0,161%**				**8,779%**

Tabela 7.1: Soluções do $PCGD_2GC$ para as instâncias $gcut1d, \ldots, gcut12d$.

O algoritmo PCGD$_2$GC encontrou soluções inteiras ótimas para 5 das 12 instâncias. Nas outras 7 instâncias, a diferença entre o valor da solução encontrada pelo PCGD$_2$GC e o valor do limite inferior fornecido pela solução de (7.2) é menor ou igual a 2. Eventualmente, algumas das soluções obtidas para estas 7 instâncias (com exceção certamente da instância $gcut4d$, para a qual encontramos uma solução melhor com o algoritmo PCGD$_2$GC^p) podem ser ótimas. De qualquer forma, todas as soluções encontradas são quase-ótimas. Na média, a diferença entre o valor da solução encontrada pelo PCGD$_2$GC e o limite inferior foi de apenas 0,161%.

Observamos ainda que o tempo gasto para resolver estas instâncias foi satisfatório, da ordem de segundos. Além disso, o ganho percentual da solução do PCGD$_2$GC em relação à solução do HFF foi de 8,779%, na média, o que é significativo.

Experimentamos também resolver as instâncias $gcut1d, \ldots, gcut12d$ com o algoritmo PCGD$_2$GC^p. A Tabela 7.2 contém os resultados obtidos.

Instância	Solução do PCGD$_2$GC^p	Limite Inferior (LI)	Diferença em relação ao LI	Tempo (seg)	Colunas Geradas	Solução do HFF	Ganho em relação ao HFF
$gcut1d$	294	294	0,000%	0,034	9	295	0,339%
$gcut2d$	345	345	0,000%	0,552	68	402	14,179%
$gcut3d$	333	332	0,301%	3,814	492	393	14,834%
$gcut4d$	837	836	0,120%	16,691	1271	977	11,429%
$gcut5d$	198	197	0,507%	0,086	25	198	0,000%
$gcut6d$	344	343	0,291%	0,400	121	418	17,308%
$gcut7d$	591	591	0,000%	1,202	136	615	4,523%
$gcut8d$	691	690	0,145%	32,757	1106	764	9,555%
$gcut9d$	131	131	0,000%	0,042	11	143	7,092%
$gcut10d$	293	293	0,000%	0,153	20	335	12,537%
$gcut11d$	331	330	0,303%	10,875	416	353	6,232%
$gcut12d$	673	672	0,149%	42,616	692	727	7,428%
Média			**0,151%**				**8,788%**

Tabela 7.2: Soluções do PCGD$_2$GC^p para as instâncias $gcut1d, \ldots, gcut12d$.

Percebemos que o desempenho do algoritmo PCGD$_2$GC^p ao resolver estas instâncias foi quase igual ao do PCGD$_2$GC. Podemos destacar algumas pequenas diferenças: para a instância $gcut4d$ foi encontrada uma solução ótima, o número de colunas geradas sofreu

um aumento de aproximadamente 40%, na média. Além disso, como já era esperado, o tempo gasto aumentou em aproximadamente 15%, na média.

7.6.2 Resolvendo Instâncias do PCGD$_2^r$

Resolvemos também as instâncias $gcut1d, \ldots, gcut12d$ permitindo rotações. Chamamos tais instâncias de $gcut1dr, \ldots, gcut12dr$. Apresentamos na Tabela 7.3 os resultados obtidos ao resolver tais instâncias com o PCGD$_2^r$GC.

Instância	Solução do PCGD$_2^r$GC	Limite Inferior (LI)	Diferença em relação ao LI	Tempo (seg)	Colunas Geradas	Solução do FFDHR	Ganho em relação ao FFDHR
$gcut1dr$	291	291	0,000%	0,052	5	291	0,000%
$gcut2dr$	283	282	0,355%	4,192	191	330	14,242%
$gcut3dr$	316	313	0,958%	6,118	323	355	10,986%
$gcut4dr$	837	836	0,120%	12,350	493	945	11,429%
$gcut5dr$	176	174	1,149%	0,366	24	200	12,000%
$gcut6dr$	302	301	0,332%	2,756	157	405	25,432%
$gcut7dr$	542	542	0,000%	4,096	121	599	9,516%
$gcut8dr$	651	650	0,153%	51,258	650	735	11,429%
$gcut9dr$	123	122	0,820%	0,472	32	140	12,143%
$gcut10dr$	270	270	0,000%	1,537	33	330	18,182%
$gcut11dr$	300	298	0,671%	43,173	249	329	8,815%
$gcut12dr$	603	601	0,333%	141,868	537	682	11,584%
Média			**0,408%**				**12,147%**

Tabela 7.3: Soluções do PCGD$_2^r$GC para as instâncias $gcut1dr, \ldots, gcut12dr$.

Observamos que o algoritmo PCGD$_2^r$GC encontrou soluções inteiras ótimas para 3 instâncias. Nas outras 9 instâncias a diferença entre o valor da solução encontrada pelo PCGD$_2$GC e o valor do limite inferior fornecido pela solução de (7.2) foi sempre menor ou igual a 3. São portanto soluções quase-ótimas. A diferença entre o valor da solução encontrada pelo PCGD$_2^r$GC e o limite inferior foi de apenas 0,408%, em média.

O tempo gasto para resolver estas instâncias também foi satisfatório, ficando sempre abaixo de 3 minutos. Comparando o valor das soluções obtidas pelo PCGD$_2^r$GC com as soluções obtidas pelo FFDHR, percebemos que houve um ganho percentual médio de 12,147%. Este ganho aumentaria para 16,168% se a comparação fosse feita com o HFF. Um fato interessante é que o ganho percentual médio do FFDHR em relação ao HFF, ao resolver as instâncias $gcut1dr, \ldots, gcut12dr$ foi de 4,774%.

Resolvemos também as instâncias $gcut1dr, \ldots, gcut12dr$ com o algoritmo PCGD$_2^r$GC^p. Apresentamos na Tabela 7.4 os resultados.

Instância	Solução do PCGD$_2^r$GC^p	Limite Inferior (LI)	Diferença em relação ao LI	Tempo (seg)	Colunas Geradas	Solução do FFDHR	Ganho em relação ao FFDHR
$gcut1dr$	291	291	0,000%	0,070	5	291	0,000%
$gcut2dr$	283	282	0,355%	5,041	252	330	14,242%
$gcut3dr$	314	313	0,319%	10,006	740	355	11,549%
$gcut4dr$	837	836	0,120%	25,042	1232	945	11,429%
$gcut5dr$	175	174	0,575%	0,537	58	200	12,500%
$gcut6dr$	301	301	0,000%	2,884	175	405	25,679%
$gcut7dr$	542	542	0,000%	4,098	121	599	9,516%
$gcut8dr$	651	650	0,153%	68,410	1154	735	11,429%
$gcut9dr$	123	122	0,820%	0,494	42	140	12,143%
$gcut10dr$	270	270	0,000%	1,546	33	330	18,182%
$gcut11dr$	299	298	0,336%	86,285	686	329	9,119%
$gcut12dr$	602	601	0,166%	181,056	945	682	11,730%
Média			**0,237%**				**12,293%**

Tabela 7.4: Soluções do PCGD$_2^r$GC^p para as instâncias $gcut1dr, \ldots, gcut12dr$.

O algoritmo PCGD$_2^r$GC^p apresentou desempenho um pouco superior ao PCGD$_2^r$GC, no que diz respeito à qualidade das soluções obtidas. A instância $gcut6dr$ foi resolvida até a otimalidade, e soluções melhores foram encontradas para as instâncias $gcut3dr$, $gcut5dr$, $gcut11dr$ e $gcut12dr$. No entanto, o preço desta melhoria foi um aumento médio de aproximadamente 97% no número de colunas geradas e de aproximadamente 44% no tempo gasto.

Observamos ainda que utilizando a idéia de perturbar as instâncias residuais, implementada pelos algoritmos PCGD_2GC^p e $\text{PCGD}_2^r\text{GC}^p$, sempre obtivemos soluções cujo valor difere do limite inferior de no máximo 1. Seria interessante resolver um grande número de instâncias e verificar se isto continua a ocorrer.

Capítulo 8

O $PCGD_2$ com Placas de Dimensões Variadas

8.1 Introdução

Podemos facilmente adaptar o método de geração de colunas aplicado ao $PCGD_2$, visto no capítulo anterior, para o problema de corte de guilhotina bidimensional com demandas e placas de dimensões variadas ($PCGD_2V$). Tal problema surge naturalmente em diversos processos de corte de guilhotina, onde sobras de cortes anteriores podem ser reaproveitados futuramente, ou ainda em situações em que o material a ser cortado é fornecido em diversas medidas.

Formalmente, o $PCGD_2V$ consiste em: dada uma lista de k tipos de placas, onde placa do tipo j tem largura L_j, altura A_j e custo C_j, e uma lista de m itens, cada item i com largura l_i, altura a_i e demanda d_i, determinar como produzir d_i unidades de cada item i, utilizando apenas cortes de guilhotina, de forma que a soma dos custos das placas utilizadas seja a menor possível [1].

Discutimos, na próxima seção, como fazer tal adaptação. Em seguida, mostramos como utilizar a mesma técnica para a variante do $PCGD_2V$ onde rotações são permitidas, que chamaremos de $PCGD_2^rV$. Discutimos brevemente como aplicar a idéia de perturbar as instâncias residuais, apresentada na Seção 7.5, ao $PCGD_2V$ e ao $PCGD_2^rV$. Finalmente, apresentamos os resultados obtidos em testes computacionais.

[1] Obviamente, para que uma instância do $PCGD_2V$ tenha solução é preciso que para todo item i ($i = 1, \ldots, m$) exista um tipo de placa j tal que $l_i \leq L_j$ e $a_i \leq A_j$.

8.2 Adaptando o PCG$_2$GC para o PCGD$_2$V

Assim como o PCGD$_2$, podemos formular o PCGD$_2$V como um problema de programação linear inteira (PLI). Para fazer isto, basta introduzir uma pequena modificação em (7.1). Observe que no PCGD$_2$ todas as placas são idênticas e portanto possuem o mesmo custo. Já no PCGD$_2$V, cada tipo de placa possui dimensões e possivelmente custos diferentes. A formulação do PLI deve incorporar esta característica. Vejamos então como alterar convenientemente (7.1).

Para cada padrão j (coluna da matriz P), seja c_j o custo da placa associada a este padrão. Se tal placa é do tipo i, então $c_j = C_i$. Representando por c o vetor cujas entradas são $c_1, \ldots, c_j$, temos então o seguinte PLI:

$$\begin{aligned} &\text{minimize} \quad c^T x \\ &Px = d \\ &x_j \geq 0 \quad \text{e inteiro} \qquad j = 1, \ldots, n. \end{aligned} \tag{8.1}$$

A relaxação linear de (8.1) é:

$$\begin{aligned} &\text{minimize} \quad c^T x \\ &Px = d \\ &x_j \geq 0 \qquad j = 1, \ldots, n. \end{aligned} \tag{8.2}$$

Podemos adaptar o algoritmo SimplexGC$_2$ para resolver (8.2). Lembremos que em cada iteração do SimplexGC$_2$ precisamos gerar um padrão cujo valor[2] seja maior que o custo da placa associada ao padrão. É possível gerar um tal padrão, se existir, fazendo no máximo k chamadas ao algoritmo PCGV$_2$PD. Em cada chamada utilizamos as dimensões de um dos k tipos de placas. O algoritmo SimplexGC$_2$V, descrito a seguir, incorpora esta modificação.

[2]O valor do padrão é a soma dos valores de cada um dos itens contidos no padrão, sendo que o valor de cada item é dado pelas variáveis duais de (8.2).

Algoritmo 8.1 SimplexGC$_2$V

Entrada: $I = (L, A, l, a, d)$ onde $L = (L_1, \ldots, L_k)$, $A = (A_1, \ldots, A_k)$, $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução ótima de (8.2):

$$\begin{aligned} &\text{minimize } c^T x \\ &Px = d \\ &x_j \geq 0 \qquad j = 1, \ldots, n. \end{aligned}$$

(onde P é a matriz dos padrões viáveis e c é o vetor de custos dos padrões)

1 Para $i = 1$ até m faça

1.1 $x = d_i$, $b_i = min(h \mid l_i \leq L_h \text{ e } a_i \leq A_h)$ e $c_i = C_{b_i}$.

2 Seja B a matriz identidade de ordem m.

3 Resolva $y^T B = \mathbb{1}^T$.

4 Para $i = 1$ até k faça

4.1 Gere uma nova coluna z executando o algoritmo PCGV$_2$PD com parâmetros L_i, A_i, l, a, y.

4.2 Se $y^T z > C_i$ vá para o passo 6.

5 Devolva B, b e x e pare (tal x corresponde apenas às colunas de B).

6 Resolva $Bw = z$.

7 Calcule $t = min(\frac{x_j}{w_j} \mid 1 \leq j \leq m,\ w_j > 0)$.

8 Calcule $s = min(j \mid 1 \leq j \leq m,\ \frac{x_j}{w_j} = t)$.

9 $b_s = i$ e $c_s = C_i$.

10 Para $i = 1$ até m faça

10.1 $B_{i,s} = z_i$.

10.2 Se $i = s$ então $x_i = t$; caso contrário, $x_i = x_i - w_i t$.

11 Retorne ao passo 3.

Na descrição do SimplexGC$_2$V, o vetor b indica a placa associada a cada coluna da matriz B. Dessa forma, as colunas de B, junto com o vetor b e os vetores *guilhotina* e *posicao*, calculados pelo PCGD$_2$, permitem construir os padrões que constituem a solução do SimplexGC$_2$V.

Dependendo dos dados de entrada, com alguns cuidados na implementação podemos obter um ganho substancial no tempo requerido pelo SimplexGC$_2$. Sejam L_{max} e A_{max} respectivamente a maior largura e a maior altura dentre todos os tipos de placas. Uma idéia que poderia ser usada é a de chamar o algoritmo PCGV$_2$PD apenas para uma placa (possivelmente fictícia) de dimensões iguais a (L_{max}, A_{max}). As matrizes V, *guilhotina* e *posicao*, calculadas pelo algoritmo, vão conter todas as soluções que seriam obtidas pelo

$PCGV_2PD$ se realizássemos k chamadas ao algoritmo utilizando as dimensões de cada um dos k tipos de placas.

Observe no entanto que a idéia descrita no parágrafo anterior nem sempre leva a uma diminuição no tempo gasto pelo algoritmo. Se houver grande diferença entre a largura e a altura de alguns dos tipos de placas, a chamada ao algoritmo $PCGV_2PD$ com as dimensões L_{max} e A_{max} pode requerer mais tempo do que k chamadas com as dimensões dos k tipos de placas.

Como exemplo, considere uma entrada do $SimplexGC_2V$ onde existe um tipo de placa com dimensões (R^2, R) e outro tipo de placa com dimensões (R, R^2). Os demais tipos de placas têm largura menor que R^2 e altura menor que R ou largura menor que R e altura menor que R^2. Uma chamada ao $PCGV_2PD$ com as dimensões L_{max} e A_{max} iria requerer tempo $\mathcal{O}(R^4)$, supondo que a quantidade de pontos de discretização seja linearmente proporcional à dimensão. Por outro lado, k chamadas com as dimensões dos k tipos de placas iria requerer tempo $\mathcal{O}(kR^3)$. Se k for muito menor do que R, o que em geral é bem razoável, é mais vantajoso não fazer uma chamada ao $PCGV_2PD$ com as dimensões L_{max} e A_{max}, mas sim k chamadas com as dimensões dos k tipos de placas.

O fato é que, numa iteração do $SimplexGC_2V$, podemos aproveitar as informações obtidas nas chamadas anteriores feitas ao $PCGV_2PD$, naquela iteração, para diminuir o tempo gasto numa nova chamada ao $PCGV_2PD$.

Observe que no $SimplexGC_2V$ utilizamos o primeiro padrão gerado pelo $PCGV_2PD$ cujo valor é maior que o custo da placa associada ao padrão. Alternativamente, poderíamos executar todas as iterações do passo 4, e dentre os k padrões gerados pelo $PCGV_2PD$ escolher aquele padrão cujo valor fosse máximo. Com isto, seria esperada uma diminuição no número de iterações do $SimplexGC_2V$, pois o valor da função objetivo tenderia a convergir mais rapidamente para o valor ótimo. Por outro lado, o tempo gasto em cada iteração aumentaria.

Em nossa implementação do $SimplexGC_2V$ utilizamos sempre o primeiro padrão gerado pelo $PCGV_2PD$ cujo valor é maior que o custo da placa associada ao padrão. Seria interessante implementar a idéia citada no parágrafo anterior, o que corresponde a usar a regra de pivotação conhecida como *largest reduced cost* [TZ93], e verificar empiricamente o que acontece com o tempo requerido pelo $SimplexGC_2V$.

Vimos como resolver (8.2). Tudo o que falta para adaptar o $PCGD_2GC$ ao $PCGD_2V$ é definir como resolver a última instância residual. Para fazer isto, utilizaremos o algoritmo HFF para empacotar os itens da instância residual nas placas que possuem o menor

custo relativo[3]. Apresentamos a seguir o algoritmo PCGD$_2$VGC, que é a adaptação do PCGD$_2$GC para o PCGD$_2$V.

Algoritmo 8.2 PCGD$_2$VGC

Entrada: Uma instância $I = (L, A, l, a, d)$ do PCGD$_2$V, onde $L = (L_1, \ldots, L_k)$, $A = (A_1, \ldots, A_k)$, $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I

1 Execute o algoritmo SimplexGC$_2$V com parâmetros L, A, l, a, d obtendo B, b e x.
2 Para $i = 1$ até m faça $x_i^* = \lfloor x_i \rfloor$.
3 Se $x_i^* > 0$ para algum i, $1 \leq i \leq m$, então
  3.1 Devolva B, b e $x_1^*, \ldots, x_m^*$ (mas não pare).
  3.2 Para $i = 1$ até m faça
    3.2.1 Para $j = 1$ até m faça $d_i = d_i - A_{i,j} x_j^*$.
  3.3 Faça $m' = 0$.
  3.4 Para $i = 1$ até m faça
    3.4.1 Se $d_i > 0$ faça $m' = m' + 1$, $l_{m'} = l_i$, $a_{m'} = a_i$ e $d_{m'} = d_i$.
  3.5 Se $m' = 0$ então pare.
  3.6 Faça $m = m'$ e volte ao passo 1.
4 $l' = \emptyset$, $a' = \emptyset$.
5 Para $i = 1$ até m faça
  5.1 Para $j = 1$ até d_i faça
    5.1.1 $l' = l' \cup \{l_i\}$, $a' = a' \cup \{a_i\}$.
6 Faça $j = min(\frac{C_i}{L_i A_i} \mid i = 1, \ldots, k)$ e $h = min(i \mid C_i = j)$.
7 Devolva a solução do algoritmo HFF executado com parâmetros L_h, A_h, l', a'.

8.3 Modificando o PCG$_2$VGC

Veremos agora como adaptar o algoritmo PCGD$_2$VGC para a variante do PCGD$_2$V em que rotações são permitidas. Chamaremos esta variante de PCGD$_2^r$V. Discutiremos ainda como utilizar a idéia de perturbar as instâncias residuais, introduzida na Seção 7.5, visando obter soluções de melhor qualidade.

[3]O custo relativo de uma placa é a razão entre seu custo e sua área.

Podemos adaptar o SimplexGC$_2$V para o caso com rotações se modificarmos os parâmetros utilizados na chamada ao PCGV$_2$PD, feita no passo 4.1 do SimplexGC$_2V$, conforme explicado na Subseção 3.3.5. Chamaremos o SimplexGC$_2$V com esta alteração de SimplexGC$_2^r$V.

Dada uma instância do PCGD$_2^r$V, vamos supor que para todo item i ($i = 1, \ldots, m$) existe um tipo de placa j tal que $l_i \leq L_j$ e $a_i \leq A_j$ (se necessário, fazemos uma rotação no item i). Com isto, garantimos que sempre será possível calcular uma solução inicial de (8.2) nos passos 1 e 2 do SimplexGC$_2^r$V.

Resolveremos a última instância residual utilizando o algoritmo FFDHR para empacotar os itens da instância residual nas placas que possuem o menor custo relativo. Dessa forma, a adaptação do algoritmo PCGD$_2$VGC para o PCGD$_2^r$V consiste em substituir as chamadas aos algoritmos SimplexGC$_2$V e HFF por chamadas aos algoritmos SimplexGC$_2^r$V e FFDHR, respectivamente. O algoritmo obtido com estas modificações será chamado de PCGD$_2^r$VGC, no entanto não apresentaremos sua descrição por ser ele muito semelhante ao PCGD$_2$VGC.

Podemos também alterar o PCGD$_2$VGC adotando a idéia de perturbar as instâncias residuais cuja solução fracionária é constituída apenas de valores menores que 1. Em tais instâncias residuais, da mesma forma que no algoritmo PCGD$_2$GC^p, utilizaremos o HFF para obter um *bom* padrão, atualizamos as demandas e, se houver demanda não atendida, retornamos ao passo 1. O algoritmo PCGD$_2$VGC^p, descrito a seguir, é o resultado desta alteração.

É possível adaptar o algoritmo PCGD$_2$GC^p para o caso com rotações simplesmente substituindo as chamadas aos algoritmos SimplexGC$_2$V e HFF por chamadas aos algoritmos SimplexGC$_2^r$V e FFDHR, respectivamente. Chamaremos de PCGD$_2^r$VGC^p o algoritmo obtido através desta alteração, mas não vamos fornecer sua descrição detalhada, pois ele é muito semelhante ao PCGD$_2$VGC^p.

Apresentamos a seguir os resultados obtidos ao aplicar os algoritmos PCGD$_2$VGC, PCGD$_2$VGC^p, PCGD$_2^r$VGC e PCGD$_2^r$VGC^p a diversas instâncias.

8.4 Resultados Computacionais

Os algoritmos foram implementados utilizando-se a linguagem C e o código executável gerado pelo compilador *gcc* versão 2.95.4 (*Debian prerelease*). Os testes executados no mesmo ambiente computacional utilizado para conduzir os experimentos relatados no

Algoritmo 8.3 PCGD$_2$VGC^p

Entrada: Uma instância $I = (L, A, l, a, d)$ do PCGD$_2$V, onde $L = (L_1, \ldots, L_k)$, $A = (A_1, \ldots, A_k)$, $l = (l_1, \ldots, l_m)$, $a = (a_1, \ldots, a_m)$ e $d = (d_1, \ldots, d_m)$.

Saída: Uma solução para I

1 Execute o algoritmo SimplexGC$_2$V com parâmetros L, A, l, a, d obtendo B, b e x.
2 Para $i = 1$ até m faça $x_i^* = \lfloor x_i \rfloor$.
3 Se $x_i^* > 0$ para algum i, $1 \leq i \leq m$, então
  3.1 Devolva B, b e $x_1^*, \ldots, x_m^*$ (mas não pare).
  3.2 Para $i = 1$ até m faça
    3.2.1 Para $j = 1$ até m faça $d_i = d_i - A_{i,j} x_j^*$.
  3.3 Faça $m' = 0$.
  3.4 Para $i = 1$ até m faça
    3.4.1 Se $d_i > 0$ faça $m' = m' + 1$, $l_{m'} = l_i$, $a_{m'} = a_i$ e $d_{m'} = d_i$.
  3.5 Se $m' = 0$ então pare.
  3.6 Faça $m = m'$ e volte ao passo 1.
4 $l' = \emptyset$, $a' = \emptyset$.
5 Para $i = 1$ até m faça
  5.1 Para $j = 1$ até d_i faça
    5.1.1 $l' = l' \cup \{l_i\}$, $a' = a' \cup \{a_i\}$.
6 Faça $j = min(\frac{C_i}{L_i A_i} \mid i = 1, \ldots, k)$ e $h = min(i \mid C_i = j)$.
7 Devolva um padrão gerado pelo algoritmo HFF, executado com parâmetros L_h, A_h, l', a', que apresente o menor desperdício de área e atualize as demandas.
8 Se houver demanda não atendida, volte ao passo 1.

capítulo anterior.

8.4.1 Resolvendo Instâncias do PCGD$_2$V

Transformamos as instâncias $gcut1, \ldots, gcut12$ em instâncias do PCGD$_2$V definindo para cada uma delas 3 tipos de placas. Chamamos tais instâncias de $gcut1dv, \ldots, gcut12dv$. Para cada tipo de placa, atribuímos custo igual à sua área. As dimensões de cada um dos tipos de placas são mostradas no Apêndice A.

Utilizamos o algoritmo PCGD$_2$VGC para resolver as instâncias $gcut1dv, \ldots, gcut12dv$. Apresentamos na Tabela 8.1 o valor da solução encontrada pelo PCGD$_2$VGC, e o limite inferior (LI) fornecido pela solução de (8.2) para o valor de uma solução inteira ótima e a

diferença percentual entre a solução do PCGD$_2$VGC e o valor do LI. Seja n a quantidade de placas utilizadas numa solução ótima de (8.2). Calculamos LI somando o valor de uma solução ótima de (8.2) a $(\lceil n \rceil - n)C_{min}$, onde C_{min} é o menor custo dentre todos os tipos de placas.

Instância	Valor da Solução	Limite Inferior (LI)	Diferença em relação ao LI	Quantidade de Placas					Tempo (seg)	Colunas Geradas
				Tipo 1	Tipo 2	Tipo 3	Total Solução	Solução (8.2)		
gcut1dv	14880000	14875313	0,032%	24	223	0	247	247	0,213	43
gcut2dv	15863750	15728072	0,863%	59	106	94	259	257	6,104	585
gcut3dv	19930000	19800108	0,656%	262	18	40	320	318	11,629	1145
gcut4dv	46477500	46278728	0,430%	462	182	108	752	749	62,437	3745
gcut5dv	42000000	41727500	0,653%	36	121	16	173	172	0,441	72
gcut6dv	74187500	74177813	0,013%	116	14	169	299	299	3,810	316
gcut7dv	123017500	122467968	0,449%	172	254	77	503	501	13,532	1063
gcut8dv	156122500	155314120	0,520%	268	195	171	634	631	109,093	2256
gcut9dv	129360000	129293847	0,051%	96	10	24	130	130	0,178	26
gcut10dv	254130000	253055471	0,425%	126	118	15	259	258	1,340	117
gcut11dv	295320000	293204167	0,722%	141	37	120	298	296	73,748	1437
gcut12dv	601450000	600245987	0,201%	316	220	75	611	610	146,726	1482
Média			**0,418%**							

Tabela 8.1: Soluções do PCGD$_2$VGC para as instâncias $gcut1dv, \ldots, gcut12dv$.

Detalhamos a utilização das placas na solução do algoritmo. Mostramos a quantidade de placas do tipo 1, do tipo 2, do tipo 3 e a quantidade total de placas utilizadas na solução do algoritmo e numa solução de (8.2) arrendondada para cima ($\lceil n \rceil$). Observe que o fato da quantidade de placas utilizadas na solução do algoritmo ser igual à $\lceil n \rceil$ não implica que a solução do algoritmo é necessariamente ótima. Lembre-se de que o nosso objetivo é minimizar a soma dos custos das placas utilizadas.

Exibimos ainda o tempo gasto e a quantidade de colunas geradas. Cada instância foi resolvida 10 vezes e os tempos apresentados foram obtidos calculando-se a média do tempo gasto nestas 10 resoluções. Não faz sentido comparar a solução encontrada pelo algoritmo com uma solução calculado pelo HFF, pois este último algoritmo não foi criado para resolver instâncias do PCGD$_2$V.

Na média, a diferença entre o valor da solução encontrada pelo PCGD$_2$GC e o limite inferior foi de apenas 0,418%. Observamos ainda que o tempo gasto para resolver estas instâncias foi sempre inferior a 3 minutos. Comparando-se o tempo gasto para resolver as instâncias $gcut1d, \ldots, gcut12d$ e as instâncias $gcut1dv, \ldots, gcut12dv$, percebemos que nestas últimas houve um aumento da ordem de aproximadamente 350%. Já o aumento

no número de colunas foi de aproximadamente 300%. Isto faz sentido se considerarmos que a quantidade de padrões viáveis (colunas da matriz P) deve ter triplicado, visto que nestas instâncias do PCGD$_2$V temos 3 tipos de placas.

Experimentamos também resolver as instâncias $gcut1dv, \ldots, gcut12dv$ com o algoritmo PCGD$_2$VGC^p. A Tabela 8.2 contém os resultados obtidos.

Instância	Valor da Solução	Limite Inferior (LI)	Diferença em relação ao LI	Quantidade de Placas					Tempo (seg)	Colunas Geradas
				Tipo 1	Tipo 2	Tipo 3	Total Solução	Solução (8.2)		
gcut1dv	14880000	14875313	0,032%	24	223	0	247	247	0,261	43
gcut2dv	15738750	15728072	0,068%	57	106	94	257	257	7,203	749
gcut3dv	19867500	19800108	0,340%	261	18	40	319	318	19,552	2128
gcut4dv	46415000	46278728	0,294%	461	182	108	751	749	102,769	7770
gcut5dv	42000000	41727500	0,653%	36	121	16	173	172	0,504	82
gcut6dv	74187500	74177813	0,013%	116	14	169	299	299	3,939	316
gcut7dv	122767500	122467968	0,245%	171	254	77	502	501	20,184	2006
gcut8dv	155872500	155314120	0,360%	267	195	171	633	631	124,928	3450
gcut9dv	129360000	129293847	0,051%	96	10	24	130	130	0,187	26
gcut10dv	254130000	253055471	0,425%	126	118	15	259	258	1,912	190
gcut11dv	295320000	293204167	0,722%	141	37	120	298	296	103,842	2551
gcut12dv	601450000	600245987	0,201%	316	220	75	611	610	152,533	1965
Média			**0,284%**							

Tabela 8.2: Soluções do PCGD$_2$VGC^p para as instâncias $gcut1dv, \ldots, gcut12dv$.

Percebemos que o desempenho do algoritmo PCGD$_2$VGC^p foi um pouco melhor, em termos de qualidade das soluções obtidas, do que o do PCGD$_2$VGC. A diferença entre o valor da solução encontrada pelo PCGD$_2$GC e o limite inferior caiu para 0,284%, na média. Por outro lado, o número de colunas geradas sofreu um aumento de aproximadamente 60%, na média, e o tempo gasto aumentou em cerca 25%, também na média.

8.4.2 Resolvendo Instâncias do PCGD$_2^r$V

Resolvemos também as instâncias $gcut1dv, \ldots, gcut12dv$ permitindo rotações. Chamamos tais instâncias de $gcut1dvr, \ldots, gcut12dvr$. Apresentamos na Tabela 8.3 os resultados obtidos ao resolver tais intâncias com o PCGD$_2^r$VGC.

Na média, a diferença entre o valor da solução encontrada pelo PCGD$_2$GC^p e o limite inferior foi de 0,631%. Comparando-se o tempo gasto para resolver as instâncias $gcut1dv, \ldots, gcut12dv$ e as instâncias $gcut1dvr, \ldots, gcut12dvr$, verificamos que o aumento nestas últimas é de aproximadamente 475%. Esta constatação está coerente com o fato de

Instância	Valor da Solução	Limite Inferior (LI)	Diferença em relação ao LI	Quantidade de Placas					Tempo (seg)	Colunas Geradas
				Tipo 1	Tipo 2	Tipo 3	Total Solução	Solução (8.2)		
gcut1dvr	13823750	13820625	0,023%	62	139	26	227	227	0,364	27
gcut2dvr	15287500	15097046	1,262%	73	55	120	248	245	10,183	417
gcut3dvr	19306875	19172691	0,700%	99	65	149	313	311	21,452	962
gcut4dvr	45046875	44595200	1,013%	327	108	293	728	721	144,937	4601
gcut5dvr	38890000	38618516	0,703%	40	75	44	159	158	2,581	122
gcut6dvr	69942500	69668304	0,394%	56	65	163	284	283	11,219	405
gcut7dvr	115385000	114610045	0,676%	215	164	90	469	466	45,930	999
gcut8dvr	152280000	151472845	0,533%	330	134	152	616	613	449,135	4405
gcut9dvr	119740000	119606667	0,111%	34	46	42	122	122	1,256	43
gcut10dvr	247660000	247422500	0,096%	37	205	14	256	256	9,228	134
gcut11dvr	284860000	281724141	1,113%	91	39	158	288	285	355,217	1621
gcut12dvr	565870000	560529066	0,953%	202	147	225	574	569	1418,728	4779
Média			**0,631%**							

Tabela 8.3: Soluções do PCGD$_2^r$VGC para as instâncias $gcut1dvr, \ldots, gcut12dvr$.

que o PCGV$_2$, usado para gerar as colunas de (8.2), requer tempo cúbico na quantidade de pontos de discretização. No caso em que rotações são permitidas, o número de tais pontos tende a dobrar. Já o aumento no número de colunas foi de aproximadamente 50%, em relação às instâncias $gcut1dv, \ldots, gcut12dv$. Isto se deve ao aumento na quantidade de padrões viáveis.

Resolvemos as instâncias $gcut1dvr, \ldots, gcut12dvr$ com o algoritmo PCGD$_2^r$VGC^p. Apresentamos na Tabela 8.4 os resultados.

O algoritmo PCGD$_2^r$VGC^p apresentou desempenho um pouco superior ao PCGD$_2^r$VGC, no que diz respeito à qualidade das soluções obtidas. Na média, a diferença entre o valor da solução encontrada pelo PCGD$_2$GC^p e o limite inferior caiu para 0,333%. No entanto, o preço desta melhoria foi um aumento médio em torno de 71% no número de colunas geradas e de aproximadamente 47% no tempo gasto.

Instância	Valor da Solução	Limite Inferior (LI)	Diferença em relação ao LI	Quantidade de Placas					Tempo (seg)	Colunas Geradas
				Tipo 1	Tipo 2	Tipo 3	Total Solução	Solução (8.2)		
*gcut*1*dvr*	13823750	13820625	0,023%	62	139	26	227	227	0,381	27
*gcut*2*dvr*	15100000	15097046	0,020%	70	55	120	245	245	12,672	609
*gcut*3*dvr*	19244375	19172691	0,374%	98	65	149	312	311	49,288	2582
*gcut*4*dvr*	44734375	44595200	0,312%	322	108	293	723	721	179,137	7517
*gcut*5*dvr*	38890000	38618516	0,703%	40	75	44	159	158	3,213	192
*gcut*6*dvr*	70192500	69668304	0,752%	57	65	163	285	283	11,219	582
*gcut*7*dvr*	114885000	114610045	0,240%	213	164	90	467	466	80,240	2191
*gcut*8*dvr*	152030000	151472845	0,368%	329	134	152	615	613	708,471	8310
*gcut*9*dvr*	119740000	119606667	0,111%	34	46	42	122	122	1,338	43
*gcut*10*dvr*	247660000	247422500	0,096%	37	205	14	256	256	9,890	134
*gcut*11*dvr*	282860000	281724141	0,403%	89	39	158	286	285	504,716	3060
*gcut*12*dvr*	563870000	560529066	0,596%	200	147	225	572	569	2079,114	9046
Média			**0,333%**							

Tabela 8.4: Soluções do $\text{PCGD}_2^r\text{VGC}^p$ para as instâncias $gcut1dvr, \ldots, gcut12dvr$.

Capítulo 9

Considerações Finais

Apresentamos neste capítulo alguns comentários finais sobre esta tese. Mencionamos nossas contribuições e tecemos algumas considerações sobre a implementação dos algoritmos por nós propostos. Discutimos também alguns possíveis desdobramentos de nossa pesquisa.

9.1 Contribuições

Nesta tese, apresentamos diversos algoritmos que desenvolvemos para problemas de corte e empacotamento bidimensional, com ênfase naquelas variantes que envolvem cortes de guilhotina. Tais algoritmos podem ser classificados, quanto à qualidade das soluções encontradas, como algoritmos de aproximação, exatos e heurísticos.

Em nossos estudos, percebemos que não existem algoritmos de aproximação propostos na literatura para o PCED_2. É verdade que existem diversos algoritmos para o caso particular do PCED_2 onde a demanda de todos os itens é igual a 1, que é chamado de PCEB_2, no entanto estes algoritmos podem requerer tempo exponencial se usados para resolver instâncias do PCED_2. Na verdade, todo algoritmo para o PCED_2 que realize pelo menos uma operação para cada retângulo empacotado vai requerer tempo exponencial, no pior caso. Assim, o projeto de um algoritmo polinomial para o PCED_2 tem que atender aos seguintes requisitos:

(a) utilizar uma quantidade polinomial de padrões (e especificar a quantidade de vezes que cada padrão deve ser utilizado).

(b) representar cada padrão com uma quantidade polinomial de memória.

Vimos que estes requisitos podem ser atendidos utilizando-se padrões homogêneos. Desenvolvemos então o algoritmo polinomial PCED_2H, que utiliza apenas padrões homogêneos, e provamos que $\text{PCED}_2\text{H}(I) \leq 4\ \text{OPT}(I) + m$, para toda instância I do PCED_2, onde m representa a quantidade de itens da instância. No entanto, podemos construir instâncias com itens muito pequenos onde $\text{PCED}_2\text{H}(I)$ pode ser tão grande quanto se queira e $\text{OPT}(I) = 1$. Disto decorre que $R^{\infty}_{\text{PCED}_2\text{H}} = \infty$.

Introduzimos então o conceito de bloco homogêneo e, baseado neste conceito, definimos o que são padrões semi-homogêneos. Desenvolvemos o algoritmo PCED_2SH, que utiliza padrões semi-homogêneos, e provamos que sua razão de aproximação absoluta é 4, e que esta razão é justa. Acreditamos ser o PCED_2H e o PCED_2SH os primeiros algoritmos de aproximação propostos para o PCED_2.

Ainda utilizando a idéia de padrões semi-homogêneos, desenvolvemos o algoritmo PEQDSH, que resolve a variante do PCED_2 na qual as placas e os itens são quadrados, que é chamada de PEQD. Provamos que o PEQDSH tem razão de aproximação assintótica entre 2,4166 e 2,6875. Até onde sabemos, este é o primeiro algoritmo de aproximação proposto para o PEQD.

Desenvolvemos ainda um algoritmo para o problema de corte de estoque bidimensional binário com rotações (PCEB^r_2), o FFDHR. Provamos que tal algoritmo possui razão de aproximação assintótica não maior que 4.

Investigamos também a aplicação de programação dinâmica ao PCGV_2. Utilizamos a fórmula de recorrência proposta por Beasley [Bea85a] e os pontos de discretização, definidos por Herz [Her72], no desenvolvimento do algoritmo PCGV_2PD, o qual encontra soluções ótimas para as instâncias do PCGV_2.

Apresentamos também os algoritmos DEE e DPD, que calculam os pontos de discretização. O primeiro é baseado em enumeração explícita, enquanto que o segundo utiliza programação dinâmica. Mostramos ainda que, se os itens não são muito pequenos em relação ao tamanho das placas[1], os algoritmos DEE e PCGV_2PD podem ser implementados de forma a requerer tempo polinomial.

Implementamos o algoritmo PCGV_2PD e resolvemos todas as instâncias do PCGV_2 encontradas na OR-LIBRARY, tendo encontrado soluções ótimas para todas essas instâncias. Vale destacar que a solução ótima de uma destas instâncias era desconhecida desde a sua proposição, há quase duas décadas. Mostramos ainda como utilizar o algoritmo PCGV_2PD para resolver instâncias do PCGV^r_2 e exibimos soluções ótimas de diversas

[1] Ou seja, $l_i > \frac{L}{k}$ e $a_i > \frac{A}{k}$ (k fixo e $i = 1, \ldots, m$).

instâncias. Em todos estes testes, o tempo requerido foi da ordem de alguns segundos.

Aplicamos o método de geração de colunas para o $PCGD_2$, utilizando o $PCGV_2PD$ para gerar as colunas, e o algoritmo HFF para resolver uma última instância residual. Esta é a idéia básica do algoritmo $PCGD_2GC$. Vimos que este algoritmo requer tempo polinomial, no caso médio, se os itens não são muito pequenos em relação ao tamanho das placas.

O algoritmo $PCGD_2GC$ reúne algoritmos de diversas naturezas, tais como o método Simplex com geração de colunas, o $PCGD_2PD$ e o HFF. Esta abordagem diversificada tem se mostrado promissora, e foi explorada em diversos trabalhos envolvendo o problema de corte unidimensional com demandas [WG96, Gau97, Cin99].

Introduzimos a idéia de perturbar as instâncias residuais como forma de obter soluções de melhor qualidade. Incorporamos esta idéia ao $PCGD_2GC$ obtendo o algoritmo $PCGD_2GC^p$. Implementamos os algoritmos $PCGD_2GC$ e $PCGD_2GC^p$, e resolvemos diversas instâncias do $PCGD_2$, tendo obtido soluções ótimas ou quase-ótimas para todas elas. O tempo requerido foi bastante satisfatório. Conforme esperávamos, o $PCGD_2GC^p$ encontrou soluções um pouco melhores que o $PCGD_2GC$ a custa de um pequeno aumento no tempo gasto.

Desenvolvemos ainda os algoritmos $PCGD_2^rGC$ e $PCGD_2^rGC^p$, que são as versões dos algoritmos $PCGD_2GC$ e $PCGD_2GC^p$, respectivamente, para a variante do $PCGD_2$ onde rotações são permitidas, que chamamos de $PCGD_2^r$. Estes algoritmos utilizam o FFDHR como sub-rotina. Vimos que, assim como o $PCGD_2GC$, o algoritmo $PCGD_2^rGC$ requer tempo polinomial, no caso médio, se os itens não são muito pequenos em relação ao tamanho das placas.

Implementamos e testamos estes dois algoritmos tendo encontrado soluções ótimas ou quase-ótimas para todas as instâncias utilizadas, em intervalos de tempo bem razoáveis. Novamente, o algoritmo que utiliza o método da perturbação das instâncias residuais, o $PCGD_2^rGC^p$, encontrou soluções de melhor qualidade, utilizando um pouco mais de tempo que o $PCGD_2^rGC$.

Estudamos ainda a variante do $PCGD_2$ na qual as placas podem não ser idênticas, chamada de $PCGD_2V$. Tal problema foi muito pouco abordado na literatura. Adaptamos o algoritmo $PCGD_2GC$ para este problema, obtendo o algoritmo $PCGD_2VGC$. Vimos que este algoritmo requer tempo polinomial, no caso médio, se os itens não são muito pequenos em relação ao tamanho das placas.

Desenvolvemos também o algoritmo $PCGD_2VGC^p$, incorporando a idéia de pertur-

bar as instâncias residuais ao $PCGD_2VGC$. Implementamos os algoritmos $PCGD_2VGC$ e $PCGD_2VGC^p$, e resolvemos diversas instâncias do $PCGD_2V$, tendo obtido soluções quase-ótimas para todas elas. Observamos que o tempo requerido foi satisfatório. O $PCGD_2VGC^p$ encontrou soluções um pouco melhores que o $PCGD_2VGC$ a custa de um aumento no tempo gasto.

Desenvolvemos ainda os algoritmos $PCGD_2^rVGC$ e $PCGD_2^rVGC^p$, que são as versões dos algoritmos $PCGD_2VGC$ e $PCGD_2VGC^p$, respectivamente, para a variante do $PCGD_2V$ onde rotações são permitidas, chamada de $PCGD_2^rV$. Implementamos e testamos estes dois algoritmos tendo encontrado soluções quase-ótimas para todas as instâncias. O tempo gasto também mostrou-se aceitável, da ordem de minutos nas instâncias mais trabalhosas. Verificamos ainda que o $PCGD_2^rVGC^p$ encontrou soluções de melhor qualidade, tendo gasto um pouco mais de tempo que o $PCGD_2^rGC$.

Verificamos que os algoritmos que propomos apresentaram um bom desempenho, em termos de tempo e de qualidade das soluções encontradas ao resolver diversas instâncias de pequeno e médio porte. Tais evidências empíricas indicam que estes algoritmos são adequados para resolver a maior parte das instâncias associadas a situações reais.

Apresentamos na Tabela 9.1 a lista dos algoritmos que propomos, classificando-os como algoritmos de aproximação, exatos ou heurísticos, e indicando sua complexidade e os problemas para os quais foram desenvolvidos.

9.2 Considerações Sobre a Implementação

As descrições fornecidas nesta tese dos algoritmos que desenvolvemos, por vezes sucintas, não deixam transparecer o esforço necessário para sua implementação. Tais algoritmos necessitam de estruturas de dados sofisticadas para que se obtenham implementações eficientes do ponto de vista de tempo de execução.

Implementamos todos os algoritmos listados na Tabela 9.1, com exceção dos algoritmos $PCED_2H$, $PCED_2SH$ e PEQDSH. Implementamos ainda os algoritmos FFD, FFDH e HFF. Desenvolvemos mais de uma dezena de programas e bibliotecas, totalizando mais de 3500 linhas de código.

Tais programas podem ser utilizados no modo interativo ou passando-se parâmetros através da linha de comando, incluindo o nome do arquivo contendo a descrição da instância a ser resolvida. Os programas fornecem uma saída de texto contendo os detalhes da solução (o nível de detalhamento pode ser selecionado).

Estes programas permitem também obter uma saída gráfica onde se pode visualizar os padrões que compõem a solução. É possível ainda gerar arquivos *PostScript* contendo tais padrões. Foi utilizando este recurso que geramos os arquivos contendo as soluções das instâncias *gcut*1, ..., *gcut*13, *gcut*1*r*, ..., *gcut*13*r* que podem ser visualizados em `http://www.ime.usp.br/~glauber/gcut`.

9.3 Pesquisas Futuras

Com respeito aos algoritmos de aproximação, diversas idéias parecem promissoras. Uma delas é modificar o algoritmo FFDHR da seguinte forma. Ao empacotar um item, podemos procurar um nível de menor altura e maior largura disponível no qual ele caiba (em alguma das orientações viáveis). Ao criar um novo nível, podemos procurar uma placa de menor altura disponível na qual ele caiba (novamente em alguma das orientações viáveis). Provalmente este algoritmo teria desempenho melhor que o FFDHR, mas talvez determinar sua razão de aproximação (justa) seja uma tarefa bastante árdua.

Outra idéia que poderia ser explorada com o objetivo de aperfeiçoar o algoritmo PEQDSH é subdividar a demanda residual de cada item de forma a obter blocos homogêneos quadrados para a instância I'. Cada um destes blocos homogêneos iria conter uma quantidade de itens que seria um quadrado perfeito. Observe que todo número inteiro n é igual à soma de k quadrados perfeitos, onde $k \leq \log n$. Isto garante que a instância I' teria tamanho polinomial em m. Poderíamos então resolvê-la com algoritmos com melhor razão de aproximação que o HFF.

Por exemplo, Kohayakawa, Miyazawa, Raghavan e Wakabayashi [KMRW01] apresentaram um algoritmo para a variante do PEQD na qual todos os itens têm demanda igual a 1. Tal algoritmo tem razão de aproximação assintótica tão próxima de 1,555... quanto se queira. Com a modificação que explicamos e utilizando o algoritmo de Kohayakawa *et al.* (ou mesmo o HFF) talvez seja possível melhorar a razão de aproximação do algoritmo PEQDSH.

Poderíamos ainda tentar melhorar a análise dos algoritmos PEQDSH e FFDHR obtendo razões de aproximação justas.

Outro desdobramento natural de nossa pesquisa seria adaptar o esquema utilizado no algoritmo PCGD_2GC para o PCED_2, no qual não há a restrição de que os cortes sejam de guilhotina. Para isto é necessário dispor de métodos para achar uma solução inicial, gerar novas colunas e resolver a última instância residual.

No caso do $PCED_2$, podemos obter uma solução inicial utilizando padrões homogêneos maximais. Para gerar as colunas poderíamos utilizar diversos algoritmos propostos na literatura para o problema de corte de estoque bidimensional com valor. Como exemplo, poderíamos citar os algoritmos propostos por Beasley [Bea85b], Arenales e Morábito [AM95], Lins, Lins e Morábito [LLM03]. Para resolver a última instância residual podemos utilizar o HFF.

Também podemos utilizar geração de colunas para a variante do $PCED_2$ onde a quantidade de itens em cada placa é limitada. Tal variante, proposta por Christofides e Whitlock, é chamada de problema de corte de estoque bidimensional restrito ($PCED_2R$). Para gerar uma solução inicial podemos utilizar padrões homogêneos (não necessariamente maximais). Cada nova coluna pode ser obtida utilizando-se um dos diversos algoritmos propostos para o problema de corte de estoque bidimensional restrito com valor [CW77, Wan83, OF90, DAD95, MA96]. A última instância residual pode ser resolvida com o HFF.

O método de geração de colunas também poderia ser aplicado para o $PCED_2R$ com a restrição de cortes de guilhotina. Note que padrões homogêneos e os padrões produzidos pelo HFF são guilhotináveis. Poderíamos gerar colunas utilizando o algoritmo de Cung, Hifi e Le Cun [CHLC00].

Um passo mais audacioso seria adaptar o método de gerações de colunas para o problema de corte de estoque tridimensional com demandas ($PCED_3$). Soluções iniciais podem ser obtidas com padrões homogêneos. Existem alguns algoritmos de aproximação para o caso tridimensional que poderiam ser usados para resolver a última instância residual [LC90b, LC92, MW97, MW00]. No entanto, não conhecemos algoritmos exatos propostos na literatura para gerar colunas. Já no caso guilhotinado do $PCED_3$, poderíamos adaptar o $PCGV_2PD$ e assim gerar novas colunas. Seria interessante descobrir se o tempo requerido seria aceitável.

Algoritmo	Tipo	Complexidade	Problemas	Comentários
PD$_2$EH	Aproximação	Polinomial	PD$_2$E	PD$_2$EH$(I) \leq 4\,\mathrm{OPT}(I) + m$
PD$_2$ESH	Aproximação	Polinomial	PD$_2$E	$R_{\mathrm{PD_2ESH}} = 4$ (razão absoluta)
PEQDSH	Aproximação	Polinomial	PEQD	$2{,}4166\ldots \leq R^{\infty}_{\mathrm{PEQDSH}} \leq 2{,}6875$ (razão assintótica)
FFDHR	Aproximação	Polinomial	PCEB$_2^r$	$R^{\infty}_{\mathrm{FFDHR}} \leq 4$ (razão assintótica)
PCGV$_2$PD	Exato	Pseudo-polinomial	PCGV$_2$ e PCGV$_2^r$	Polinomial para itens grandes
DEE	Exato	Exponencial	Pontos de discretização	Polinomial para itens grandes
DPD	Exato	Pseudo-polinomial	Pontos de discretização	
PCGD$_2$GC	Heurístico	Exponencial	PCGD$_2$	Polinomial, no caso médio, para itens grandes
PCGD$_2$GC^p	Heurístico	Exponencial	PCGD$_2$	
PCGD$_2^r$GC	Heurístico	Exponencial	PCGD$_2^r$	Polinomial, no caso médio, para itens grandes
PCGD$_2^r$GC^p	Heurístico	Exponencial	PCGD$_2^r$	
PCGD$_2$VGC	Heurístico	Exponencial	PCGD$_2$V	Polinomial, no caso médio, para itens grandes
PCGD$_2$VGC^p	Heurístico	Exponencial	PCGD$_2$V	
PCGD$_2^r$VGC	Heurístico	Exponencial	PCGD$_2^r$V	Polinomial, no caso médio, para itens grandes
PCGD$_2^r$VGC^p	Heurístico	Exponencial	PCGD$_2^r$V	

Tabela 9.1: Algoritmos propostos nesta tes

Apêndice A

Instâncias de Teste

Apresentamos aqui a descrição das instâncias utilizadas nos testes computacionais cujos resultados foram relatados nos capítulos 6, 7 e 8. Tais instâncias têm como base as instâncias do $PCGV_2$ disponíveis na OR-LIBRARY[1], denominadas de $gcut1, \ldots, gcut13$. Tal biblioteca é uma coleção de instâncias de testes para uma grande variedade de problemas na área de pesquisa operacional. Uma descrição desta biblioteca e de seus objetivos é dada em [Bea90].

Fornecemos também os valores das soluções encontradas para estas instâncias de testes e no caso das instâncias do $PCGD_2$ e do $PCGD_2V$ exibimos um limite inferior para o valor de uma solução ótima.

A.1 Instâncias de teste para o $PCGD_2$

No capítulo 6 apresentamos os testes realizados com o algoritmo $PCGD_2PD$, nos quais foram utilizadas as instâncias $gcut1, \ldots, gcut13$. Nestas instâncias, cada item tem valor igual à sua área. Os demais dados que compõem estas instâncias são fornecidos nas tabelas A.1 e A.2.

Experimentamos também resolver as instâncias $gcut1, \ldots, gcut13$ permitindo rotações. Tais instâncias foram chamadas de $gcut1r, \ldots, gcut13r$, e seus dados são idênticos aos das instâncias $gcut1, \ldots, gcut13$.

[1] `http://mscmga.ms.ic.ac.uk/info.html`

A.2 Instâncias de Teste para o $PCGD_2$

Os testes realizados com os algoritmos $PCGD_2GC$ e $PCGD_2GC^p$, discutidos no Capítulo 7, foram feitos com as instâncias $gcut1d, \ldots, gcut12d$. Estas instâncias foram obtidas a partir das instâncias $gcut1, \ldots, gcut12$, atribuindo-se a cada item uma demanda gerada aleatoriamente entre 1 e 100. Apresentamos na Tabela A.3 as demandas dos itens nas instâncias $gcut1d, \ldots, gcut12d$. Os outros dados que compõem estas instâncias são idênticos aos das instâncias $gcut1, \ldots, gcut12$, e podem ser obtidos nas tabelas A.1 e A.2.

Com o intuito de testar os algoritmos $PCGD_2^rGC$ e $PCGD_2^rGC^p$, experimentamos também resolver as instâncias $gcut1d, \ldots, gcut12d$ permitindo rotações. Tais instâncias foram chamadas de $gcut1dr, \ldots, gcut12dr$, e seus dados são idênticos aos das instâncias $gcut1d, \ldots, gcut12d$.

A.3 Instâncias de Teste para o $PCGD_2V$

Testamos os algoritmos $PCGD_2VGC$ e $PCGD_2VGC^p$, propostos no Capítulo 8, com as instâncias $gcut1dv, \ldots, gcut12dv$. Tais instâncias foram obtidas a partir das instâncias $gcut1d, \ldots, gcut12d$, definindo-se três tipos de placas para cada uma delas. Cada tipo de placa têm custo igual à sua área. Apresentamos na Tabela A.4 as larguras e alturas dos tipos de placas nas instâncias $gcut1dv, \ldots, gcut12dv$. Os outros dados que compõem estas instâncias são idênticos aos das instâncias $gcut1d, \ldots, gcut12d$, e podem ser obtidos nas tabelas A.1, A.2 e A.3.

Para testar os algoritmos $PCGD_2^rVGC$ e $PCGD_2^rVGC^p$, também propostos no Capítulo 8, resolvemos as instâncias $gcut1dv$, ..., $gcut12dv$ permitindo rotações. Chamamos tais instâncias de $gcut1dvr$, ..., $gcut12dvr$, e seus dados são os mesmos das instâncias $gcut1dv, \ldots, gcut12dv$.

A.4 Soluções das Instâncias de Teste

Na Tabela A.5 apresentamos os valores das soluções encontradas para as instâncias de testes descritas nas tabelas A.1, A.2, A.3 e A.4. Todas as instâncias de teste do $PCGV_2$ foram resolvidas até a otimalidade. No caso das instâncias do $PCGD_2$ e do $PCGD_2V$ fornecemos também um limite inferior para o valor de uma solução ótima.

gcut1	
$m = 10$	
$L = 250$	
$A = 250$	
l	a
167	184
114	118
167	152
83	140
70	86
143	166
120	160
66	148
87	141
69	165

gcut5	
$m = 10$	
$L = 500$	
$A = 500$	
l	a
198	205
179	155
364	236
272	147
352	145
343	245
132	174
164	250
282	356
342	151

gcut9	
$m = 10$	
$L = 1000$	
$A = 1000$	
l	a
310	426
673	468
426	463
325	498
555	540
292	455
343	341
362	491
305	688
459	607

gcut2	
$m = 20$	
$L = 250$	
$A = 250$	
l	a
120	133
135	186
86	75
103	73
66	85
135	97
91	175
131	176
71	176
153	72
87	148
168	107
118	90
140	109
132	159
152	93
135	68
121	158
68	94
155	76

gcut6	
$m = 20$	
$L = 500$	
$A = 500$	
l	a
313	305
293	222
330	253
212	256
132	189
149	296
294	137
225	345
345	220
337	177
189	300
234	321
335	272
354	244
149	169
355	165
260	220
210	241
194	372
287	134

gcut10	
$m = 20$	
$L = 1000$	
$A = 1000$	
l	a
730	300
269	717
463	369
642	464
453	329
455	667
506	482
560	362
483	260
693	345
381	510
456	586
457	453
707	658
639	650
691	359
434	700
576	291
728	739
545	742

gcut3	
$m = 30$	
$L = 250$	
$A = 250$	
l	a
66	80
164	107
64	184
121	86
163	135
85	98
81	102
103	186
152	106
176	139
111	118
69	169
146	133
112	112
133	160
63	129
163	152
110	155
96	136
92	142
84	143
119	133
71	71
146	84
93	86
89	86
101	146
172	73
73	169
99	99

gcut7	
$m = 30$	
$L = 500$	
$A = 500$	
l	a
348	220
255	184
191	249
212	194
348	322
171	369
206	327
221	277
250	202
205	226
193	280
364	311
264	224
244	347
270	338
224	236
168	177
285	347
315	294
247	219
315	302
129	147
268	273
350	352
186	258
365	374
145	259
284	184
129	198
146	351

gcut11	
$m = 30$	
$L = 1000$	
$A = 1000$	
l	a
541	745
344	301
413	294
266	341
543	388
367	701
526	707
286	706
614	289
348	592
673	431
475	362
562	439
530	638
609	375
540	274
486	634
272	377
623	306
266	714
341	336
340	296
531	479
571	589
674	496
560	497
509	389
650	332
474	285
419	503

Tabela A.1: Instâncias *gcut*1, *gcut*2, *gcut*3, *gcut*5, *gcut*6, *gcut*7, *gcut*9, *gcut*10 e *gcut*11.

gcut4		*gcut8*		*gcut12*		*gcut13*	
$m = 50$		$m = 50$		$m = 50$		$m = 32$	
$L = 250$		$L = 500$		$L = 1000$		$L = 3000$	
$A = 250$		$A = 500$		$A = 1000$		$A = 3000$	
l	a	l	a	l	a	l	a
139	103	277	242	572	665	365	185
166	99	233	177	482	640	378	200
88	74	182	216	264	594	410	165
174	139	147	134	349	566	425	148
128	109	213	127	276	660	425	296
175	179	315	212	745	422	439	116
62	82	261	175	434	622	464	1106
178	87	210	281	446	433	520	205
121	77	243	256	668	506	520	350
125	97	159	314	405	635	540	530
73	84	217	147	486	320	549	1413
69	97	260	349	554	549	549	1882
122	159	222	199	392	491	553	496
66	69	296	127	459	528	555	755
111	118	263	302	426	579	555	496
165	86	226	250	728	359	555	659
71	83	264	344	644	465	567	473
163	162	237	179	382	323	572	592
67	132	245	137	268	352	572	975
121	152	137	328	324	512	572	1175
68	150	321	154	277	426	572	1575
138	124	200	174	254	685	572	1390
141	105	270	128	483	432	572	1490
183	122	299	179	678	414	572	1590
171	63	165	370	546	590	572	1690
101	64	198	182	716	640	572	1890
103	95	155	295	542	400	610	625
110	107	278	219	634	596	660	490
184	158	184	155	497	696	690	447
85	97	187	270	417	300	949	445
86	92	208	293	290	605	949	478
127	155	261	204	321	520	970	463
92	147	344	168	543	290		
106	97	213	135	599	269		
177	108	362	162	467	679		
164	150	362	374	583	421		
68	84	237	322	729	723		
62	151	281	308	465	590		
123	134	155	344	614	585		
158	178	210	163	446	327		
70	140	325	225	734	616		
173	117	360	225	479	606		
147	136	346	347	656	278		
141	145	131	144	537	355		
84	164	284	284	604	519		
144	141	262	228	746	596		
92	103	146	178	439	281		
156	98	254	164	339	405		
160	111	332	238	273	526		
127	131	313	304	625	568		

Tabela A.2: Instâncias *gcut*4, *gcut*8, *gcut*12 e *gcut*13.

gcut1d	*gcut5d*	*gcut9d*	*gcut3d*	*gcut7d*	*gcut11d*	*gcut4d*	*gcut8d*	*gcut12d*
90	91	91	33	37	44	77	83	95
75	71	61	75	61	32	75	51	3
71	94	40	94	62	99	16	30	58
13	74	95	65	47	9	17	19	23
39	32	18	2	80	38	64	29	58
79	63	30	78	30	33	78	97	35
63	60	56	49	42	28	35	24	100
93	27	95	4	6	9	15	85	24
61	58	51	7	97	76	53	36	1
85	75	53	6	74	10	27	74	66
			5	50	40	79	87	3
			20	45	96	76	18	3
gcut2d	*gcut6d*	*gcut10d*	26	78	81	60	47	20
61	64	68	55	72	5	7	69	91
75	66	47	50	89	69	23	23	22
33	78	69	64	19	29	87	13	64
39	60	2	25	47	89	19	54	25
20	6	78	80	80	80	62	62	61
79	46	82	90	34	24	35	42	58
56	51	42	84	87	94	68	40	83
49	16	52	95	46	47	61	45	14
34	27	13	33	52	90	81	46	76
96	74	32	64	89	38	36	11	60
19	82	8	5	23	60	83	13	73
92	9	43	68	86	22	75	72	66
8	43	12	30	20	1	53	3	4
79	23	12	100	42	27	8	12	19
13	73	93	33	58	7	78	85	1
2	72	12	62	43	5	84	53	90
78	93	21	87	30	16	27	32	43
40	40	39				96	44	41
17	80	6				8	8	8
92	61	99				41	37	29
						27	19	3
						31	10	68
						48	84	57
						11	64	69
						79	58	16
						90	68	24
						63	51	26
						56	72	4
						40	71	34
						78	92	21
						86	82	72
						67	70	75
						75	10	80
						94	40	31
						50	10	32
						93	42	40
						59	76	12

Tabela A.3: Demandas das instâncias $gcut1d, \ldots, gcut12d$.

	Tipo 1		Tipo 2		Tipo 3	
Instância	Largura	Altura	Largura	Altura	Largura	Altura
gcut1dv	200	300	250	250	225	275
gcut2dv	200	300	250	250	225	275
gcut3dv	200	300	250	250	225	275
gcut4dv	200	300	250	250	225	275
gcut5dv	400	600	500	500	450	550
gcut6dv	400	600	500	500	450	550
gcut7dv	400	600	500	500	450	550
gcut8dv	400	600	500	500	450	550
gcut9dv	800	1200	1000	1000	900	1100
gcut10dv	800	1200	1000	1000	900	1100
gcut11dv	800	1200	1000	1000	900	1100
gcut12dv	800	1200	1000	1000	900	1100

Tabela A.4: Larguras e alturas dos tipos de placas nas instâncias $gcut1dv$, ..., $gcut12vd$.

Instância	Solução do $PCGV_2PD$
gcut1	56460
gcut2	60536
gcut3	61036
gcut4	61698
gcut5	246000
gcut6	238998
gcut7	242567
gcut8	246633
gcut9	971100
gcut10	982025
gcut11	980096
gcut12	979986
gcut13	8997780

Instância	Solução do $PCGV_2PD$
gcut1r	58136
gcut2r	60611
gcut3r	61626
gcut4r	62265
gcut5r	246000
gcut6r	240951
gcut7r	245866
gcut8r	247787
gcut9r	971100
gcut10r	982025
gcut11r	980096
gcut12r	979986
gcut13r	9000000

Instância	Solução do $PCGD_2GC$	Solução do $PCGD_2GC^p$	Limite Inferior
gcut1d	294	294	294
gcut2d	345	345	345
gcut3d	333	333	332
gcut4d	838	837	836
gcut5d	198	198	197
gcut6d	344	344	343
gcut7d	591	591	591
gcut8d	691	691	690
gcut9d	131	131	131
gcut10d	293	293	293
gcut11d	331	331	330
gcut12d	673	673	672

Instância	Solução do $PCGD_2^rGC$	Solução do $PCGD_2^rGC^p$	Limite Inferior
gcut1dr	291	291	291
gcut2dr	283	283	282
gcut3dr	316	314	313
gcut4dr	837	837	836
gcut5dr	176	175	174
gcut6dr	302	301	301
gcut7dr	542	542	542
gcut8dr	651	651	650
gcut9dr	123	123	122
gcut10dr	270	270	270
gcut11dr	300	299	298
gcut12dr	603	602	601

Instância	Solução do $PCGD_2VGC$	Solução do $PCGD_2VGC^p$	Limite Inferior
gcut1dv	14880000	14880000	14875313
gcut2dv	15863750	15738750	15728072
gcut3dv	19930000	19867500	19800108
gcut4dv	46477500	46415000	46278728
gcut5dv	42000000	42000000	41727500
gcut6dv	74187500	74187500	74177813
gcut7dv	123017500	122767500	122467968
gcut8dv	156122500	155872500	155314120
gcut9dv	129360000	129360000	129293847
gcut10dv	254130000	254130000	253055471
gcut11dv	295320000	295320000	293204167
gcut12dv	601450000	601450000	600245987

Instância	Solução do $PCGD_2^rVGC$	Solução do $PCGD_2^rVGC^p$	Limite Inferior
gcut1dvr	13823750	13823750	13820625
gcut2dvr	15287500	15100000	15097046
gcut3dvr	19306875	19244375	19172691
gcut4dvr	45046875	44734375	44595200
gcut5dvr	38890000	38890000	38618516
gcut6dvr	69942500	70192500	69668304
gcut7dvr	115385000	114885000	114610045
gcut8dvr	152280000	152030000	151472845
gcut9dvr	119740000	119740000	119606667
gcut10dvr	247660000	247660000	247422500
gcut11dvr	284860000	282860000	281724141
gcut12dvr	565870000	563870000	560529066

Tabela A.5: Valor das soluções encontradas para as instâncias de teste.

Referências Bibliográficas

[AM95] M. Arenales and R. Morabito, *An and/or-graph approach to the solution of two-dimensional non-guillotine cutting problems*, European Journal of Operations Research **84** (1995), 599–617. Citado na(s) página(s) 120

[AMT84] I. Adler, N. Megiddo, and M. J. Todd, *New results on the average behavior of simplex algorithms*, Bull. Amer. Math. Soc. (N.S.) **11** (1984), no. 2, 378–382. Citado na(s) página(s) 89

[Bak85] B. S. Baker, *A new proof for the first-fit decreasing bin-packing algorithm*, Journal of Algorithms **6** (1985), 49–70. Citado na(s) página(s) 32

[BBK81] B. S. Baker, D. J. Brown, and H. P. Katseff, *A $\frac{5}{4}$ algorithm for two-dimensional packing*, Journal of Algorithms **2** (1981), 348–368. Citado na(s) página(s) 38, 44

[BBK82] D. J. Brown, B. S. Baker, and H. P. Katseff, *Lower bounds for the on-line two dimensional packing algorithms*, Acta Informatica **18** (1982), 207–225. Citado na(s) página(s) 5, 39

[BC81] B. S. Baker and E. G. Coffman Jr., *A tight asymptotic bound for next fit decreasing bin packing*, SIAM J. Algebraic Discrete Methods **2** (1981), 147–152. Citado na(s) página(s) 44

[Bea85a] J. E. Beasley, *Algorithms for unconstrained two-dimensional guillotine cutting*, Journal of the Operational Research Society **36** (1985), no. 4, 297–306. Citado na(s) página(s) 7, 23, 67, 77, 116

[Bea85b] J. E. Beasley, *An exact two-dimensional nonguillotine cutting tree search procedure*, Oper. Res. **33** (1985), no. 1, 49–64. Citado na(s) página(s) 120

[Bea90] ———, *Or-library: distributing test problems by electronic mail*, Journal of the Operational Research Society **41** (1990), no. 11, 1069–1072. Citado na(s) página(s) 77, 123

[BMR95] V. Bafna, S. Muthukrishnan, and R. Ravi, *Computing similarity between RNA strings*, Combinatorial Pattern Matching (Espoo, 1995), Lecture Notes in Comput. Sci., vol. 937, Springer, Berlin, 1995, pp. 1–16. Citado na(s) página(s) 66

[Bor90] Karl-Heinz Borgwardt, *Probabilistic analysis of the simplex method*, Mathematical developments arising from linear programming (Brunswick, ME, 1988), Contemp. Math., vol. 114, Amer. Math. Soc., Providence, RI, 1990, pp. 21–34. Citado na(s) página(s) 89

[BRSL97] J. Bramel, W. T. Rhee, and D. Simchi-Levi, *Average-case analysis of the bin-packing problem with general cost structures*, Naval Res. Logist. **44** (1997), no. 7, 673–686. Citado na(s) página(s) 33

[BvW96] D. Blitz, A. van Vliet, and G. J. Woeginger, *Lower bounds on the asymptotic worst-case ratio of online bin packing algorithms*, Unpublished manuscript (1996). Citado na(s) página(s) 5, 42

[Cap02] A. Caprara, *Packing 2-dimensional bins in harmony*, Proceedings of the 43-rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, 2002, pp. 490–499. Citado na(s) página(s) 41, 42, 44

[CCG+02] E. G. Coffman, Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, and M. Yannakakis, *Perfect packing theorems and the average-case behavior of optimal and online bin packing*, SIAM Rev. **44** (2002), no. 1, 95–108 (electronic), Revised reprint of SIAM J. Discrete Math. **13** (2000), no. 3, 384–402. Citado na(s) página(s) 43

[CFC94] C. H. Cheng, B. R. Feiring, and T. C. E. Cheng, *The cutting stock problem – a survey*, International Journal of Production Economics **36** (1994), no. 3, 291–305. Citado na(s) página(s) 29

[CGF+86] J. Csirik, G. Galambos, J. B. Frenk, A. M. Frieze, and A. H. G. Rinnooy Kan, *A probabilistic analysis of the next fit decreasing bin packing heuristic*, Oper. Res. Lett. **5** (1986), 233–236. Citado na(s) página(s) 43

[CGJ78] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, *An application of bin-packing to multiprocessor scheduling*, SIAM Journal on Computing **7** (1978), 1–17. Citado na(s) página(s) 14

[CGJ82] F. R. K. Chung, M. R. Garey, and D. S. Johnson, *On packing two-dimensional bins*, SIAM J. Algebraic Discrete Methods **3** (1982), 66–76. Citado na(s) página(s) 10, 40, 41, 44

[CGJ97] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms*, ch. Approximation algorithms for bin packing - a survey, PWS (ed. D. Hochbaum), 1997. Citado na(s) página(s) 29, 34, 44

[CGJT80] E. G. Coffman Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan, *Performance bounds for level oriented two-dimensional packing algorithms*, SIAM Journal on Computing **9** (1980), 808–826. Citado na(s) página(s) 35, 36, 38, 44

[CHLC00] Van-Dat Cung, Mhand Hifi, and Bertrand Le Cun, *Constrained two-dimensional cutting stock problems a best-first branch-and-bound algorithm*, Int. Trans. Oper. Res. **7** (2000), no. 3, 185–210. Citado na(s) página(s) 120

[Chv80] V. Chvátal, *Linear programming*, W. H. Freeman and Company, New York, 1980. Citado na(s) página(s) 9, 85

[Cin98] G. F. Cintra, *Algoritmos híbridos para problemas de corte unidimensional*, Master's thesis, Instituto de Matemática e Estatística, São Paulo, 1998. Citado na(s) página(s) 89

[Cin99] ———, *Algoritmos híbridos para o problema de corte unidimensional*, XXV Conferência Latinoamericana de Informática (Assunção), 1999. Citado na(s) página(s) 88, 89, 117

[CJSW97] E. G. Coffman, Jr., D. S. Johnson, P. W. Shor, and R. R. Weber, *Bin packing with discrete item sizes. II. Tight bounds on first fit*, Random Structures Algorithms **10** (1997), no. 1–2, 69–101, Average-case analysis of algorithms (Dagstuhl, 1995). Citado na(s) página(s) 43

[CLM02] A. Caprara, A. Lodi, and M Monacci, *An approximation scheme for the two-stage, two-dimensional bin packing problem*, Proceedings of the Ninth Conference on Integer Programming and Combinatorial Optimization (IPCO'02) (A.S. Schulz W.J. Cook, ed.), Springer-Verlag, 2002, pp. 320–334. Citado na(s) página(s) 42

[CLR88] E. G. Coffman Jr., G. S. Lueker, and A. H. G. Rinooy Kan, *Asymptotic methods in the probabilistic analysis of sequencing and packing heuristics*, Manage. Sci. **34** (1988), no. 3, 266–290. Citado na(s) página(s) 43

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms*, second ed., MIT Press, Cambridge, MA, 2001. Citado na(s) página(s) 9, 66

[CR89] D. Coppersmith and P. Raghavan, *Multidimensional on-line bin packing: Algorithms and worst-case analysis*, Oper. Res. Lett. **8** (1989), no. 1, 17–20. Citado na(s) página(s) 42

[CV93] J. Csirik and A. Van Vliet, *An on-line algorithm for multidimensional bin packing*, Operations Research Letters **13** (1993), 149–158. Citado na(s) página(s) 42

[CW77] N. Christofides and C. Whitlock, *An algorithm for two dimensional cutting problems*, Operations Research **25** (1977), 30–44. Citado na(s) página(s) 27, 120

[CW97] János Csirik and Gerhard J. Woeginger, *Shelf algorithms for on-line strip packing*, Information Processing Letters **63** (1997), no. 4, 171–175. Citado na(s) página(s) 5, 39, 44

[DAD95] V. P. Daza, A. G. Alvarenga, and J. Diego, *Exact solutions for constrained two-dimensional cutting problems*, European Journal of Operations Research **84** (1995), 633–644. Citado na(s) página(s) 27, 120

[Dan57] G. B. Dantzig, *Discrete-variable extremum problems*, Operations Research **5** (1957), 266–277. Citado na(s) página(s) 14

[DF92] H. Dyckhoff and U. Finke, *Cutting and packing in production and distribution.*, Springer-Verlag, Heidelberg, 1992. Citado na(s) página(s) 29

[DKAG85] H. Dyckhoff, H. J. Kruse, D. Abel, and T. Gal, *Trim loss and related problems*, Omega **13** (1985), 59–72. Citado na(s) página(s) 15

[Dow85] W. B. Dowsland, *Two and three dimensional packing problems and solution methods*, New Zealand Journal of Operational Research **13** (1985), 1–18. Citado na(s) página(s) 15

[Dyc90] H. Dyckhoff, *A typology of cutting and packing problems*, European Journal of Operations Research **44** (1990), 145–159. Citado na(s) página(s) 6, 13

[EC71] S. Eilon and N. Christofides, *The loading problems*, Management Science **17** (1971), 259–268. Citado na(s) página(s) 14

[Fal96] E. Falkenauer, *A hybrid grouping genetic algorithm for bin packing*, Journal of Heuristics **2** (1996), 5–30. Citado na(s) página(s) 33

[FG87] J. B. Frenk and G. Galambos, *Hybrid next fit algorithm for the two-dimensional rectangle bin packing problem*, Computing **39** (1987), 201–217. Citado na(s) página(s) 42, 44

[FL81] W. Fernandez de la Vega and G. S. Lueker, *Bin packing can be solved within 1+ϵ in linear time*, Combinatorica **1** (1981), 349–355. Citado na(s) página(s) 3, 34, 42, 44

[FMC+01] C. G. Fernandes, F. K. Miyazawa, M. Cerioli, P. Feofiloff, and et al., *Uma introdução sucinta a algoritmos de aproximação*, Publicações Matemáticas do IMPA. [IMPA Mathematical Publications], Instituto de Matemática Pura e Aplicada (IMPA), Rio de Janeiro, 2001, 23º Colóquio Brasileiro de Matemática. [23rd Brazilian Mathematics Colloquium]. Citado na(s) página(s) 10

[FMW99] C.E. Ferreira, F.K. Miyazawa, and Y. Wakabayashi, *Packing squares into squares*, Pesquisa Operacional, **19** (1999), no. 2, 223–237. Citado na(s) página(s) 4

[Fri98] Erich Friedman, *Packing unit squares in squares: a survey and new results*, Electron. J. Combin. **5** (1998), no. 1, Dynamic Survey 7, 24 pp. (electronic). Citado na(s) página(s) 29

[FvR97] Awi Federgruen and Garrett van Ryzin, *Probabilistic analysis of a generalized bin packing problem and applications*, Oper. Res. **45** (1997), no. 4, 596–609. Citado na(s) página(s) 43

[Gau97] T. Gau, *Solution methods for the standard one-dimensional cutting stock problem*, Physica, Heidelberg, 1997. Citado na(s) página(s) 117

[GG61] P. Gilmore and R. Gomory, *A linear programming approach to the cutting stock problem*, Operations Research **9** (1961), 849–859. Citado na(s) página(s) 81

[GG63] _____, *A linear programming approach to the cutting stock problem - part II*, Operations Research **11** (1963), 863–888. Citado na(s) página(s) 81

[GG65] _____, *Multistage cutting stock problems of two and more dimensions*, Operations Research **13** (1965), 94–120. Citado na(s) página(s) 7, 66, 67

[GG66] _____, *The theory and computation of knapsack functions*, Operations Research **14** (1966), 1045–1075. Citado na(s) página(s) 81

[GGJ78] M. R. Garey, R. L. Graham, and D. S. Johnson, *On a number theoretic bin packing conjecture*, Proc. 5th Hungarian Combinatorics Colloquium (Amsterdam), North-Holland, 1978, pp. 377–392. Citado na(s) página(s) 32

[GGJY76] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao, *Resource constrained scheduling as generalized bin packing*, J. Combinatorial Theory Ser. A **21** (1976), 257–298. Citado na(s) página(s) 32, 44

[GJ75] M. R. Garey and D. S. Johnson, *Complexity results for multiprocessor scheduling under resource constraints*, SIAM J. Comput. **4** (1975), no. 4, 397–411. Citado na(s) página(s) 4

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of $\mathcal{NP}$-completeness*, W. H. Freeman and Co., San Francisco, 1979. Citado na(s) página(s) 5, 9, 10, 11, 33

[GJ81] _____, *Approximation algorithms for bin packing problems: a survey*, in: D. Ausiello and M. Lucertini (eds.), Analysis and Design of Algorithms in Combinatorial Optimization, CISM Courses and Lectures **266** (1981), 147–172. Citado na(s) página(s) 14

[Gol81] Igal Golan, *Performance bounds for orthogonal oriented two-dimensional packing algorithms*, SIAM J. Comput. **10** (1981), no. 3, 571–582. Citado na(s) página(s) 38, 44

[GW95] G. Galambos and G. Woeginger, *On-line bin packing—a restricted survey.*, ZOR—Math. Methods Oper. Res. **42** (1995), 25–45. Citado na(s) página(s) 29

[Her72] J. C. Herz, *A recursive computational procedure for two-dimensional stock-cutting*, IBM Journal of Research Development (1972), 462–469. Citado na(s) página(s) 7, 67, 70, 116

[Hif01] M. Hifi, *Exact algorithms for large-scale unconstrained two and three staged cutting problems*, Comput. Optim. Appl. **18** (2001), no. 1, 63–88. Citado na(s) página(s) 25

[HT90] R. W. Haessler and F. B. Talbot, *Load planning for shipments of low density products*, European Journal of Operations Research **44** (1990), no. 2, 289–299. Citado na(s) página(s) 15

[JDU+74] D. S. Johnson, A. Demars, J. D. Ullman, M. R. Garey, and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM Journal on Computing **3** (1974), 299–325. Citado na(s) página(s) 32, 34, 44

[JG85] D. S. Johnson and M. R. Garey, *A $\frac{71}{60}$ theorem for bin packing*, J. Complexity **1** (1985), 65–106. Citado na(s) página(s) 44

[Joh73] D. S. Johnson, *Near-optimal bin packing algorithms*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1973. Citado na(s) página(s) 30, 31, 32, 34, 44

[Joh74] ———, *Fast algorithms for bin packing*, J. Comput. Syst. Sci. **8** (1974), 272–314. Citado na(s) página(s) 3, 34

[KK82] N. Karmarkar and R. M. Karp, *An efficient approximation scheme for the one dimensional bin packing problem*, Proceedings, 23rd Ann. Symp. on Foundations of Computer Science (Los Angeles), IEEE Computer Society, 1982, pp. 312–320. Citado na(s) página(s) 3, 34, 44

[KM72] Victor Klee and George J. Minty, *How good is the simplex algorithm?*, Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin) (New York), Academic Press, 1972, pp. 159–175. Citado na(s) página(s) 88

[KMRW01] Y. Kohayakawa, F.K. Miyazawa, P. Raghavan, and Y. Wakabayashi, *Multidimensional cube packing*, Brasilian Symposium on Graphs and Combinatorics. Electronic Notes of Discrete Mathematics (GRACO'2001), Elsevier Science, 2001. Citado na(s) página(s) 42, 119

[KR00] Claire Kenyon and Eric Rémila, *A near-optimal solution to a two-dimensional cutting stock problem*, Math. Oper. Res. **25** (2000), no. 4, 645–656. Citado na(s) página(s) 39, 42, 44

[KSS75] K. L. Krause, Y. Y. Shen, and H. D. Schwetman, *Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems*, J. Association Comput. Mach. **22** (1975), 522–550. Citado na(s) página(s) 34

[LC90a] K. Li and K-H. Cheng, *A generalized harmonic algorithm for on-line multidimensional bin packing*, TR UH-CS-90-2, University of Houston, January 1990. Citado na(s) página(s) 42

[LC90b] ———, *On three-dimensional packing*, SIAM Journal on Computing **19** (1990), 847–867. Citado na(s) página(s) 120

[LC92] ———, *Heuristic algorithms for on-line packing in three dimensions*, Journal of Algorithms **13** (1992), 589–605. Citado na(s) página(s) 120

[LLM03] Lauro Lins, Sóstenes Lins, and Reinaldo Morabito, *An l-approach for packing (l,w)-rectangles into rectangular and l-shaped pieces*, Journal of the Operational Research Society **54** (2003), 777–789. Citado na(s) página(s) 120

[LMM02] Andrea Lodi, Silvano Martello, and Michele Monaci, *Two-dimensional packing problems: a survey*, European J. Oper. Res. **141** (2002), no. 2, 241–252, Cutting and packing. Citado na(s) página(s) 29

[LS55] J. Lorie and L. J. Savage, *Three problems in capital rationing*, Journal of Business **28** (1955), 229–239. Citado na(s) página(s) 14

[LTW+90] J. Y-T. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. L. Chin, *Packing squares into a square*, J. Parallel and Distributed Computing **10** (1990), 271–275. Citado na(s) página(s) 4

[Lue75] G. S. Lueker, *Two NP-complete problems in nonnegative integer programming*, Report no. 178, Princeton University, Computer Science Laboratory, Princeton, NJ, 1975. Citado na(s) página(s) 4

[MA96] R. Morabito and M. N. Arenales, *Staged and constrained two-dimensional guilhotine cutting problems: an and/or-graph approach*, European Journal of Operational Research **94** (1996), 548–560. Citado na(s) página(s) 120

[Miy97] F. K. Miyazawa, *Algoritmos de aproximação para problemas de empacotamento*, Ph.D. thesis, Instituto de Matemática e Estatística, São Paulo, 1997. Citado na(s) página(s) 40, 42, 44

[MT80] S. Martello and P. Toth, *Optimal and canonical solutions of the change making problem*, European Journal of Operations Research **4** (1980), 322–329. Citado na(s) página(s) 14

[MW97] F. K. Miyazawa and Y. Wakabayashi, *An algorithm for the three-dimensional packing problem with asymptotic performance analysis*, Algorithmica **18** (1997), no. 1, 122–144. Citado na(s) página(s) 120

[MW00] ———, *Approximation algorithms for the orthogonal z-oriented three-dimensional packing problem*, SIAM Journal on Computing **29** (2000), no. 3, 1008–1029. Citado na(s) página(s) 120

[NST98] Christoph Nitsche, Guntram Scheithauer, and Johannes Terno, *New cases of the cutting stock problem having MIRUP*, Math. Methods Oper. Res. **48** (1998), no. 1, 105–115. Citado na(s) página(s) 89

[OF90] J. F. Oliveira and J. S. Ferreira, *An improved version of Wang's algorithm for two-dimensional cutting problems*, European Journal of Operations Research **44** (1990), 256–266. Citado na(s) página(s) 27, 120

[OMW84] H. L. Ong, M. J. Magazine, and T. S. Wee, *Probabilistic analysis of bin packing heuristics*, Operations Research **32** (1984), 983–998. Citado na(s) página(s) 43

[Ram92] B. Ram, *The pallet loading problem: A survey*, International Journal of Production Economics **28** (1992), 217–225. Citado na(s) página(s) 29

[Ric91] Michael B. Richey, *Improved bounds for harmonic-based bin packing algorithms*, Discrete Appl. Math. **34** (1991), no. 1-3, 203–227, Combinatorics and theoretical computer science (Washington, DC, 1989). Citado na(s) página(s) 34

[RST95] J. Riehme, G. Scheithauer, and J. Terno, *The solution of two-stage guillotine cutting stock problems having extremly varying order demands*, TR MATH-NM-05-1995, TU Dresden, 1995. Citado na(s) página(s) 25

[RST02] Jürgen Rietz, Guntram Scheithauer, and Johannes Terno, *Tighter bounds for the gap and non-IRUP constructions in the one-dimensional cutting stock problem*, Optimization **51** (2002), no. 6, 927–963. Citado na(s) página(s) 89

[Sch94] Ingo Schiermeyer, *Reverse-Fit: a 2-optimal algorithm for packing rectangles*, Algorithms—ESA '94 (Utrecht), Lecture Notes in Comput. Sci., vol. 855, Springer, Berlin, 1994, pp. 290–299. Citado na(s) página(s) 39

[SL94] David Simchi-Levi, *New worst-case results for the bin-packing problem*, Naval Res. Logist. **41** (1994), no. 4, 579–585. Citado na(s) página(s) 32, 33, 34

[SP92] P. E. Sweeney and E. R. Paternoster, *Cutting and packing problems: a categorized, application-oriented research bibliography*, Journal of the Operational Research Society **43** (1992), 691–706. Citado na(s) página(s) 29

[ST92] G. Scheithauer and J. Terno, *About the gap between the optimal values of the integer and continuous relaxation one-dimensional cutting stock problem*, Operations Research Proceedings (Berlin), Springer-Verlag, 1992. Citado na(s) página(s) 89

[ST95] ———, *The modified integer round-up property of the one-dimensional cutting stock problem*, European Journal of Operations Research **84** (1995), 562–571. Citado na(s) página(s) 89

[ST97] Guntram Scheithauer and Johannes Terno, *Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem*, Oper. Res. Lett. **20** (1997), no. 2, 93–100. Citado na(s) página(s) 89

[Ste83] F. Stehling, *Der Bedarf an Münzen in verschiedenen Münzsystemen*, Methods of Operations Research **46** (1983), 509–521. Citado na(s) página(s) 14

[Ste97] A. Steinberg, *A strip-packing algorithm with absolute performance bound 2*, SIAM Journal on Computing **26** (1997), no. 2, 401–409. Citado na(s) página(s) 39

[SW97] P. Schwerin and G. Wäscher, *The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP*, International Transactions in Operational Research **4** (1997), 377–389. Citado na(s) página(s) 33

[TZ93] Tamás Terlaky and Shu Zhong Zhang, *Pivot rules for linear programming: a survey on recent theoretical developments*, Ann. Oper. Res. **46/47** (1993), no. 1-4, 203–233, Degeneracy in optimization problems. Citado na(s) página(s) 106

[Vli92] A. Van Vliet, *An improved lower bound for online bin packing algorithms*, Inform. Process. Lett. **43** (1992), 277–284. Citado na(s) página(s) 5, 34

[Wan83] P. Y. Wang, *Two algorithms for constrained two dimensional cutting stock problems*, Operations Research **31** (1983), 573–586. Citado na(s) página(s) 27, 120

[WCO00] Larry Wall, Tom Christiansen, and Jon Orwant, *Programming perl*, 3rd ed., O'Reilly & Associates, July 2000. Citado na(s) página(s) 97

[WG93] G. Wäscher and T. Gau, *Two approaches to the cutting stock problem*, IFORS'93 Conference (Lisbon), 1993. Citado na(s) página(s) 89

[WG96] ———, *Heuristics for the integer one-dimensional cutting stock problem: a computational study*, OR Spektrum **18** (1996), 131–144. Citado na(s) página(s) 89, 117

[WM82] T. S. Wee and M. J. Magazine, *Assembly line balancing as generalized bin packing*, Oper. Res. Lett. **1** (1982), 56–58. Citado na(s) página(s) 14

[Wäs90] G. Wäscher, *An LP-based approach to cutting stock problems with multiple objectives*, European Journal of Operations Research **44** (1990), 175–184. Citado na(s) página(s) 19

[Xpr02] Xpress, *Xpress optimizer reference manual*, DASH Optimization, Inc, 2002. Citado na(s) página(s) 96

[Yue91] M. Yue, *A simple proof of the inequality* $\mathrm{FFD}(L) \leq \frac{11}{9}\mathrm{OPT}(L) + 1$, $\forall L$ *for the FFD bin-packing algorithm*, Acta Math. Appl. Sin., Engl. Ser. **4** (1991), 321–331. Citado na(s) página(s) 33, 44

[YZH91] Horácio Yanasse, Alan Zinober, and Reginald Harris, *Two-dimensional cutting stock with multiple stock sizes*, J. Oper. Res. Soc. **42** (1991), no. 8, 673–683. Citado na(s) página(s) 27

[Zad73] Norman Zadeh, *A bad network problem for the simplex method and other minimum cost flow algorithms*, Math. Programming **5** (1973), 255–266. Citado na(s) página(s) 88

Índice Remissivo

Sigla	Problema
$PCEB_2$	Problema de corte de estoque bidimensional binário
$PCED_2$	Problema de corte de estoque bidimensional com demandas
$PCEV_2$	Problema de corte de estoque bidimensional com valor
$PCGB_2$	Problema de corte de guilhotina bidimensional binário
$PCGB_2^r$	Problema de corte de guilhotina bidimensional binário e com rotações
$PCGD_2$	Problema de corte de guilhotina bidimensional com demandas
$PCGD_2^r$	Problema de corte de guilhotina bidimensional com demandas e com rotações
$PCGD_2V$	Problema de corte de guilhotina bidimensional com demandas e placas de dimensões variadas
$PCGD_2^rV$	Problema de corte de guilhotina bidimensional com demandas, placas de dimensões variadas e com rotações
$PCGV_2$	Problema de corte de guilhotina bidimensional com valor
$PCGV_2^r$	Problema de corte de guilhotina bidimensional com valor e com rotações
PEBB	Problema de empacotamento de *bins* binário
PEFB	Problema de empacotamento em faixa binário
PEQB	Problema de empacotamento de quadrados em quadrados binário
PEQD	Problema de empacotamento de quadrados em quadrados com demandas

Tabela A.6: Principais problemas estudados nesta tese e suas siglas.