Algoritmos para Problemas de Empacotamento

Eduardo Candido Xavier

Tese de Doutorado

Instituto de Computação
Universidade Estadual de Campinas

Algoritmos para Problemas de Empacotamento

Eduardo Candido Xavier[1]

5 de dezembro de 2006

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa
  Instituto de Computação, Unicamp (Orientador)
- Prof. Dra. Yoshiko Wakabayashi
  Instituto de Matemática e Estatística, USP
- Prof. Dr. Horácio Hideki Yanasse
  Laboratório Associado de Computação e Matemática Aplicada, INPE
- Prof. Dr. Cid Carvalho de Souza
  Instituto de Computação, Unicamp
- Prof. Dr. Orlando Lee
  Instituto de Computação, Unicamp

[1] **Auxílio financeiro da Capes e do CNPq**

Substitua pela ficha catalográfica

Algoritmos para Problemas de Empacotamento

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Eduardo Candido Xavier e aprovada pela Banca Examinadora.

Campinas, 5 de dezembro de 2006.

Prof. Dr. Flávio Keidi Miyazawa
Instituto de Computação, Unicamp (Orientador)

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

Substitua pela folha com a assinatura da banca

Resumo

Neste trabalho estudamos diversos problemas de empacotamento considerados NP-difíceis. Assumindo a hipótese de que $\mathrm{P} \neq \mathrm{NP}$, sabemos que não existem algoritmos eficientes (complexidade de tempo polinomial) exatos para resolver tais problemas. Uma das abordagens consideradas para tratar tais problemas é a de algoritmos de aproximação, que são algoritmos eficientes e que geram soluções com garantia de qualidade. Neste trabalho apresentamos alguns algoritmos aproximados para problemas de empacotamento com aplicações práticas. Outra maneira de se lidar com problemas NP-difíceis é o desenvolvimento de heurísticas. Neste trabalho também apresentamos heurísticas baseadas no método de geração de colunas para problemas de corte e empacotamento bidimensional. Resultados computacionais sugerem que tais heurísticas são eficientes e geram soluções de muito boa qualidade.

Abstract

In this work we study several packing problems that are NP-hard. If we consider that $\mathrm{P} \neq \mathrm{NP}$, we know that there are no efficient (polynomial time complexity) exact algorithms to solve these problems. One way to deal with these kind of problems is to use approximation algorithms, that are efficient algorithms that produce solutions with quality guarantee. We present several approximation algorithms for some packing problems that have practical applications. Another way to deal with NP-hard problems is to develop heuristics. We also consider column generation based heuristics for packing problems. In this case, we present column generation algorithms for some two dimensional packing problems and also present computational tests with the proposed algorithms. The computational results shows that the heuristics are efficient and produce solutions of very good quality.

À meus pais, João e Dita,
meus irmãos Jose, Alexandre e César
e à Silvana.

Agradecimentos

Hoje considero a educação uma das coisas mais importantes da minha vida. Passei quase vinte e dois anos da minha vida freqüentando algum tipo de escola. Vinte e dois anos tentando fazer com que eu ficasse educado, e eu ainda estou tentando. Pena que agora não me resta outra coisa a não ser estudar por conta própria. Considerando a importância que dou à educação, gostaria de agradecer a todos aqueles que a influenciaram. Todos os professores do ensino básico e médio, professores da graduação na UFPR e da pós-graduação na UNICAMP. Agradecimentos especiais vão para meu orientador de iniciação científica, Alexandre Direne e ao Renato Carmo por me introduzir à Teoria da Computação. Também gostaria de agradecer aos excelentes professores que tive aqui na UNICAMP, e em especial a alguns, que na minha opinião estão entre os melhores professores que tive em minha vida: Célia de Mello, Cid de Souza, Flávio Miyazawa, Ricardo Dahab e Orlando Lee.

Eu tive a felicidade de nascer em uma família que valoriza a educação, e meus mais sinceros agradecimentos vão para os meus pais João e Dita, por terem me proporcionado o melhor que podiam, e aos meus irmãos (Alexandre, Zinho e César), que por serem mais velhos do que eu, sempre tinham alguma coisa para me ensinar. Quando era criança me recordo do Zinho, do César e do Sandre, várias vezes me ensinando alguma coisa de matemática ou física, ou alguma outra coisa que tinha dúvidas.

Algumas pessoas quando terminam o doutorado acabam por receber um doutorado duplo. O primeiro é outorgado pela instituição onde estudou, e é dado na área de estudo. O outro é outorgado pela própria pessoa, e é em arrogância. Tive a sorte de ter um orientador que recusou o segundo título. Por isso agradeço ao Flávio Miyazawa pelos quase 6 anos de orientação produtiva e tranqüila.

Agradeço à todos colegas e amigos do IC por terem tornado a pós-graduação mais divertida. Não vou citar nomes pra não esquecer de alguém, mas se você está lendo isso, estes agradecimentos vão para você.

Agradeço ao pessoal do Laboratório LOCO, e aos “irmãos” de orientação: André (Piazão), Carlos (Louco), Evandro, Mamão (Rafael), Meira, Nilton (O Cara), e Victor.

Agradeço especialmente à minha noiva Silvana, pelo carinho, amor e amizade durante todos estes anos.

Agradeço finalmente à toda população brasileira, especialmente à mais pobre que contribuiu de forma involuntária com as bolsas da CAPES e do CNPq que proporcionaram com que eu fizesse este trabalho.

Sabiá que anda atrás de João de Barro vira servente de pedreiro.

(Autor deconhecido)

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Neste trabalho apresentamos algoritmos voltados para problemas de empacotamento. Muitas das variações de problemas de empacotamento são problemas de otimização que pertencem à classe NP-difícil. Problemas de otimização, na sua forma geral, têm como objetivo maximizar ou minimizar uma função definida sobre um certo domínio. A teoria clássica de otimização trata do caso em que o domínio é infinito. Já no caso dos chamados problemas de otimização combinatória, o domínio é tipicamente finito; além disso, em geral é fácil listar os seus elementos e também testar se um dado elemento pertence a esse domínio. Ainda assim, a idéia ingênua de testar todos os elementos deste domínio na busca pelo melhor mostra-se inviável na prática, mesmo para instâncias de tamanho moderado.

Como trabalho de doutorado fizemos um estudo na área de otimização combinatória, mais especificamente sobre problemas de empacotamento. Quando nos referimos a problemas de empacotamento, estamos tratando de uma grande classe de problemas onde temos um ou mais objetos grandes n-dimensionais, os quais chamamos de recipientes, e vários objetos menores também n-dimensionais os quais chamamos de itens. O nosso objetivo é empacotar itens dentro de recipientes maximizando ou minimizando uma dada função objetivo. Provavelmente os dois problemas de empacotamento mais conhecidos sejam o problema de empacotamento unidimensional (*Bin Packing Problem*) e o problema da mochila (*Knapsack Problem*). No primeiro problema temos uma lista de itens e um número infinito de recipientes iguais. O objetivo é empacotar todos os itens no menor número de recipientes possível. No segundo problema, temos um único recipiente e uma lista de itens, cada item com um determinado valor. O objetivo do problema é empacotar itens da lista que maximizem a soma de seus valores.

Neste trabalho, assumimos a hipótese de que $\mathrm{P} \neq \mathrm{NP}$. Desta forma, tais problemas e muitos outros problemas de otimização que são NP-difíceis não possuem algoritmos eficientes exatos. Muitos destes problemas aparecem em aplicações práticas e há um forte apelo econômico para resolvê-los. Problemas de empacotamento possuem aplicações em diversas áreas da Computação e Pesquisa Operacional. Podemos citar como exemplos, problemas em aloca-

ção de recursos em computadores ou problemas clássicos de corte de materiais em indústrias [37, 42, 36, 17, 18, 19, 20, 21, 16].

Como não conseguimos resolver tais problemas de forma exata e eficiente, buscamos alternativas que possam ser úteis. Existem vários métodos que são muito utilizados na prática como o uso de heurísticas, programação inteira, métodos híbridos, redes neurais, algoritmos genéticos, dentre outros.

O foco deste trabalho de doutorado está no desenvolvimento de heurísticas para problemas de empacotamento, principalmente aquelas em que conseguimos estabelecer uma razão, no pior caso, entre a solução devolvida pela heurística e a solução ótima. Tais heurísticas são comumente chamadas de algoritmos de aproximação. Neste caso, o algoritmo sacrifica a otimalidade em troca da garantia de uma solução aproximada computável em tempo polinomial em relação ao tamanho da entrada. Em linhas gerais, algoritmos de aproximação são aqueles que não necessariamente produzem uma solução ótima, mas soluções que estão dentro de um certo fator da solução ótima. Esta garantia deve ser satisfeita para todas as instâncias do problema. Desta forma, devemos dar uma demonstração formal deste fato. Também é nosso interesse o estudo de heurísticas baseadas no método de geração de colunas. Muitos problemas de empacotamento podem ser formulados com programas lineares que possuem um número muito grande de colunas. Desta forma, a resolução de tais programas lineares por métodos tradicionais se torna impraticável. Muitos destes programas lineares fornecem soluções fracionárias muito próximas das soluções inteiras. Com isso, há um grande interesse em resolver tais sistemas lineares, e usá-los para obter soluções inteiras. Como o número de colunas é muito grande aplica-se o método de geração de colunas.

1.1 Objetivos do Trabalho

O principal objetivo deste trabalho é apresentar novos algoritmos para alguns problemas de empacotamento. Para cada problema considerado também buscamos apresentar aplicações práticas destes, de forma a se ter uma maior motivação por parte do leitor. Para alguns destes problemas fizemos inclusive testes computacionais, demonstrando a aplicabilidade prática de alguns dos algoritmos propostos.

1.2 Organização do Texto

Esta tese está organizada como uma coletânea de artigos. Uma das grandes vantagens desta forma de apresentação é mostrar de forma direta os resultados obtidos na tese de doutorado. Por outro lado, uma desvantagem é que pode haver repetições de definições durante o texto. De qualquer maneira optamos por esta forma de organização. Cada artigo apresenta aplica-

ções dos problemas considerados, bem como os algoritmos propostos. A seguir detalhamos a organização do texto.

No Capítulo 2 apresentamos algumas definições e conceitos básicos que são usados nos capítulos seguintes.

No Capítulo 3 fazemos um resumo dos principais resultados desta tese que correspondem aos resultados dos capítulos seguintes.

Do Capítulo 4 até o Capítulo 8 apresentamos cinco artigos com os principais resultados desta tese.

No Capítulo 9 apresentamos as conclusões e trabalhos futuros.

Capítulo 2

Preliminares

Este capítulo contém, de forma resumida, definições e noções básicas que serão necessárias no decorrer da leitura do trabalho. Primeiramente apresentamos conceitos e problemas básicos de empacotamento. Em seguida, introduzimos definições básicas sobre algoritmos de aproximação, e discutimos brevemente algumas técnicas usadas no desenvolvimento de algoritmos aproximados. Apresentamos ainda um resultado de análise combinatória que é utilizado frequentemente neste trabalho e por fim apresentamos o algoritmo *simplex* e como ele pode ser usado com o método de geração de colunas.

2.1 Problemas de Empacotamento

Nesta seção descrevemos os principais problemas de empatotamento tratados nesta tese.

Nos problemas de empacotamento temos um ou mais objetos grandes n-dimensionais, os quais chamamos de *recipientes*, e vários objetos menores também n-dimensionais os quais chamamos de *itens*. O nosso objetivo é empacotar itens dentro de recipientes, de forma a maximizar ou minimizar uma dada função objetivo. No caso geral, tanto os itens quanto os recipientes podem assumir qualquer forma, ou modelo (retângulos, esferas, formas quaisquer etc.). O empacotamento deve ser feito de tal maneira que os itens não ocupem um mesmo espaço e que as capacidades do recipiente sejam respeitadas.

Uma tipologia para vários tipos de problemas de empacotamento foi feita por Dyckhoff [14] e mais recentemente uma nova tipologia foi proposta por Wäscher *et al.* [41].

Problemas de empacotamento são muito comuns na indústria e em problemas computacionais. Em muitas aplicações, faz-se necessário cortar materiais (placas de metal, vidro, papel ou tecido) em itens menores para se atender uma determinada demanda. Note que para decidirmos como cortar o material, podemos supor que estamos empacotando os itens no material. Como exemplos de problemas computacionais, citamos problemas de alocação de tarefas em computadores. Tais problemas podem ser vistos como problemas de empacotamento. Temos

várias tarefas, que correspondem a itens, e devemos alocá-las a um conjunto de processadores de modo a otimizar uma certa função objetivo, como por exemplo, maximizar o peso das tarefas que podem ser processadas até um determinado momento. Neste caso, o problema consiste em empacotar itens unidimensionais, que possuem um tamanho e um peso, em recipientes cujo tamanho é dado pelo limite de tempo.

Problemas de empacotamento são comuns em nosso cotidiano. O famoso astrônomo Johannes Kepler, em 1611, já se perguntava a melhor maneira de alocação de esferas. Um feirante que deseja empilhar laranjas por exemplo, pode se perguntar qual a melhor maneira de realizar tal tarefa. O problema de empacotamento de esferas proposto por Kepler só teve uma solução confirmada formalmente em 1998 [38], em uma sequência de trabalhos feitos por Hales [23].

Definiremos agora três problemas de empacotamento básicos que são tratados nesta tese. Estes problemas serão tratados com algumas restrições extras que são apresentadas em cada capítulo.

O primeiro problema, chamado *bin packing*, tem como entrada uma lista de itens $L = (a_1, \ldots, a_m)$, cada item com tamanho $s(a_i)$, e um número B que indica o tamanho de um recipiente. Assumimos que para todo item $a_i \in L$, vale que $s(a_i) \leq B$. Este problema consiste em empacotar todos os itens de L no menor número possível de recipientes, ou seja, devemos achar uma partição $P_1, \ldots, P_q$ de L tal que q seja mínimo e $\sum_{a_i \in P_j} s(a_i) \leq B$, para cada parte P_j.

Existem as versões em outras dimensões deste problema, como *bin packing* bidimensional, tridimensional etc. Nestes casos, os recipientes e os itens possuem tamanhos dados por uma tupla que indica o seu tamanho em cada dimensão.

Podemos assumir ainda que cada item $a_i \in L$ possui uma multiplicidade d_i. Neste caso devemos gerar um empacotamento que contém d_i itens do *tipo* a_i, $i = 1, \ldots, m$. Os problemas com multiplicidade são conhecidos na literatura como *cutting stock*.

Detalhes sobre algoritmos de aproximação para este problema podem ser encontrados em Coffman *et al.* [10]. Para o caso bidimensional pode-se consultar as resenhas de Lodi *et al.* [29, 30].

O segundo problema é conhecido como *knapsack*, ou problema da mochila. Neste caso temos apenas um recipiente de tamanho B, e uma lista de itens $L = (a_1, \ldots, a_m)$ cada item com tamanho $s(a_i)$ e valor $p(a_i)$. O objetivo do problema é empacotar um subconjunto dos itens de L em um recipiente de tamanho B de tal forma que a soma dos valores destes itens empacotados seja maximizada. Também podemos considerar as versões multi-dimensionais deste problema.

O problema, como foi definido, é conhecido como restrito, ou mochila 0/1. Neste caso, cada item pode ser empacotado apenas uma vez. Na versão não-restrita, um item $a_i \in L$ pode ser empacotado várias vezes. Um esquema de aproximação para o problema restrito foi proposto na década de 70 por Ibarra e Kim [26]. Maiores informações sobre este problema podem ser

encontradas no livro de Martello e Toth [31].

O terceiro problema é conhecido como *strip packing*. Neste problema temos uma lista de itens bidimensionais $L = (a_1, \ldots, a_m)$, cada item a_i com tamanho $(x(a_i), y(a_i))$, e uma faixa de largura L e altura infinita. O objetivo do problema é empacotar todos os itens na faixa de tal maneira que seja minimizada a altura total utilizada para empacotar os itens. Versões multidimensionais podem ser consideradas.

Dentre os diversos algoritmos propostos para este problema, destacamos um esquema de aproximação apresentado por Kenyon e Rémila [27, 28], e algoritmos exatos como o proposto por Martello *et al.* [32].

Neste trabalho, consideramos duas classes de algoritmos para problemas de empacotamento, a classe *online* e a classe *offline*. Os algoritmos chamados *offline*, são aqueles onde todos os dados da instância são conhecidos pelo algoritmo de antemão. Na classe de algoritmos chamados *online*, os dados da instância não são conhecidos de antemão pelo algoritmo. Itens chegam com o passar do tempo e devem ser empacotados assim que estiverem disponíveis. Muitos problemas de empacotamento têm esta característica, e neste caso é preciso desenvolver algoritmos *online* para estes problemas. Um exemplo de problema *online* é o *Bin Packing Online*, que tem a mesma definição que o problema *offline*, com exceção de que os itens a serem empacotados chegam um por vez, de tal forma que um algoritmo para este problema deve empacotar um item sem saber quais os próximos itens da lista.

2.2 Algoritmos de Aproximação

Nesta seção apresentamos a notação utilizada e alguns conceitos básicos sobre algoritmos de aproximação.

Dado um algoritmo $\mathcal{A}$, para um problema de minimização, se I for uma instância para este problema, denotamos por $\mathcal{A}(I)$ o valor da solução devolvida pelo algoritmo $\mathcal{A}$ aplicado à instância I. Denotamos por $\mathrm{OPT}(I)$ o correspondente valor para uma solução ótima de I. Dizemos que um algoritmo $\mathcal{A}$ tem um *fator de aproximação* α, ou é α*-aproximado*, se $\mathcal{A}(I)/\mathrm{OPT}(I) \leq \alpha$, para toda instância I. No caso dos algoritmos probabilísticos, consideramos a desigualdade $E[\mathcal{A}(I)]/\mathrm{OPT}(I) \leq \alpha$, onde a esperança $E[\mathcal{A}(I)]$ é tomada sobre todas as escolhas aleatórias feitas pelo algoritmo. Estes limites de desempenho são chamados de absolutos. Em problemas de empacotamento é comum considerar aproximações *assintóticas*. Neste caso dizemos que um algoritmo $\mathcal{A}$ tem fator de aproximação assíntótico α se $\lim_{OPT(I)\to\infty} \mathcal{A}(I)/\mathrm{OPT}(I) \leq \alpha$. É importante ressaltar que algoritmos de aproximação considerados neste trabalho têm complexidade de tempo polinomial.

Do ponto de vista teórico, os algoritmos de aproximação mais desejados são aqueles que obtêm valores mais próximos possível do valor ótimo. Dado um valor constante $\epsilon > 0$, é possível mostrar para vários problemas, que estes admitem algoritmos com fatores de aproximação

$(1+\epsilon)$, no caso de problemas de minimização, e $(1-\epsilon)$, no caso de problemas de maximização, onde ϵ pode ser tomado tão pequeno quanto se queira. Chamamos estes algoritmos de *esquemas de aproximação polinomial* ou PTAS (*Polynomial Time Approximation Scheme*) se apresentarem tais fatores de aproximação e tempo de execução polinomial na entrada. Chamamos de FPTAS (*Fully Polynomial Time Approximation Scheme*) o esquema de aproximação que tem tempo de execução polinomial na entrada e em $\frac{1}{\epsilon}$. Logo, dentre os dois tipos, os algoritmos mais desejados são os FPTAS.

Em problemas de empacotamento é também comum considerar esquemas de aproximação assintóticos. A definição é parecida com a que demos anteriormente para aproximação assíntotica, mas neste caso deve valer a desigualdade $\lim_{OPT(I)\to\infty} \mathcal{A}(I)/\mathrm{OPT}(I) \leq (1+\epsilon)$. Denotamos por APTAS (*Asymptotic Polynomial Time Approximation Scheme*) os algoritmos que apresentarem tais fatores de aproximação e tempo de execução polinomial na entrada. Denotamos por AFPTAS (*Asymptotic Fully Polynomial Time Approximation Scheme*) os APTAS que têm tempo de execução polinomial na entrada e em $\frac{1}{\epsilon}$.

Uma outra forma de análise de problemas NP-difíceis é a utilização do conceito de aproximação dual proposto por Hochbaum e Shmoys [24]. Neste caso um algoritmo é dual aproximado se ele consegue encontrar uma solução, não necessariamente viável, cujo valor é no máximo o valor de uma solução ótima. Neste caso, a medida de qualidade de aproximação está ligada a quão inviável é a solução. Existem algumas situações na prática nas quais as restrições de viabilidade são flexíveis e o conceito de algoritmos de aproximação duais podem ser utilizados. Neste caso o fator de aproximação é uma razão entre alguma inviabilidade da solução em relação à restrição de viabilidade. Por exemplo, no caso do problema *bin packing*, onde todos os recipientes têm tamanho 1, um algoritmo que para qualquer instância I do problema encontra uma solução que usa $OPT(I)$ recipientes de tamanho 1.3 corresponde a um algoritmo dual aproximado com fator de aproximação 1.3.

No caso de algoritmos *online* o termo comumente utilizado para designar fator de aproximação é *competitive ratio*, que é a razão, no pior caso, entre o valor da solução devolvida pelo algoritmo sobre o valor de uma solução ótima para a versão *offline* do problema. No texto, quando tratarmos de problemas *online*, usaremos o termo fator de aproximação com o mesmo significado de *competitive ratio*.

Ao projetar um algoritmo aproximado e provar que o mesmo tem um certo fator de aproximação α, é interessante verificar se este fator de aproximação α demonstrado é o melhor possível. Para isto, devemos encontrar uma instância cuja razão entre a solução obtida pelo algoritmo e sua solução ótima é igual, ou tão próxima quanto se queira, de α. Neste caso, dizemos que o fator de aproximação α do algoritmo é justo.

Nos últimos anos surgiram várias técnicas de caráter geral para o desenvolvimento de algoritmos de aproximação. Algumas destas são: *arredondamento de soluções via programação linear, dualidade em programação linear e método primal-dual, algoritmos probabilísticos (e*

sua desaleatorização) e programação semidefinida (veja [15, 25, 39, 5]). Além disso, resultados sobre *provas verificáveis probabilisticamente* [2, 3, 4] permitiram obter vários resultados sobre a impossibilidade de aproximações.

Uma estratégia comum para se tratar problemas de otimização combinatória é formular o problema como um programa linear inteiro e resolver a relaxação linear deste, uma vez que isto pode ser feito em tempo polinomial. Programação linear tem sido usado para a obtenção de algoritmos aproximados através de diversas maneiras. Uma muito comum é o uso de arredondamentos das soluções fracionárias do programa linear. Outra técnica é resolver o sistema dual do programa linear, em vez do primal, e em seguida obter uma solução com base nas variáveis duais. Outra técnica mais recente, é o uso do método de aproximação primal-dual, que tem sido usado para obter diversos algoritmos combinatórios usando a teoria de dualidade em programação linear. Neste caso, o método é em geral combinatório, não requerendo a resolução de programas lineares e consiste de uma generalização do método primal-dual tradicional.

Já no caso de algoritmos probabilísticos, o algoritmo contém passos que dependem de uma seqüência de bits aleatórios. Neste caso, a análise da solução gerada pelo algoritmo é calculada com base no valor esperado da solução. É interessante observar que apesar do modelo parecer restrito, a maioria dos algoritmos probabilísticos pode ser desaleatorizada, através do método das esperanças condicionais, tornando-se algoritmos determinísticos (veja [15, 5]). A versão probabilística é, em geral, mais simples de se implementar e mais fácil de se analisar que a correspondente versão determinística. Além disso, muitos dos algoritmos de aproximação combinam o uso de técnicas de programação linear com técnicas usadas em algoritmos probabilísticos, considerando o valor das variáveis obtidas pela relaxação linear como probabilidades.

No caso da técnica de programação semidefinida, temos um sistema de programação matemática para o problema, que não precisa ser estritamente linear. Em alguns casos é possível ter restrições não lineares na formulação, como por exemplo restrições quadráticas. Se a formulação for escrita sob certas condições, o problema pode ser resolvido em tempo polinomial. A vantagem deste método é que muitos problemas podem ser representados através de modelos de programação semidefinida, isto é, formulações não necessariamente lineares. Goemans e Williansom [22] apresentaram uma forma bastante inovadora de se arredondar as soluções de um sistema quadrático, através do arredondamento probabilístico, considerando cada uma das variáveis do sistema como um vetor na esfera unitária.

Estas técnicas, tanto isoladamente como em conjunto, têm sido usadas nos últimos anos com sucesso em diversos problemas de otimização combinatória.

Outro tópico importante em algoritmos de aproximação é a inaproximalidade de problemas. Dado um certo problema Q, dizemos que este problema possui fator de inaproximalidade α, se não existir um algoritmo α-aproximado para Q. Uma das maneiras para se demonstrar tais resultados, é mostrar que se existir um algoritmo α-aproximado para um problema Q, então

podemos resolver em tempo polinomial um problema Q' que seja NP-difícil. Resultados importantes nesta área foram feitos com a utilização de provas verificáveis probabilisticamente, devido a Arora *et al.* [3, 4]. Para mais detalhes sobre resultados de inaproximalidade veja [2, 5, 25, 39].

2.3 Um Resultado sobre Contagem

Nesta seção apresentamos um resultado sobre contagem que é utilizado em diversas partes desta tese. Problemas de contagem aparecem como um ramo da análise combinatória onde busca-se descobrir uma expressão que determina a quantidade de elementos de um determinado conjunto. Uma visão mais ampla sobre problemas de contagem e análise combinatória pode ser encontrada em [13].

Considere os seguintes problemas:

1. Qual o número de soluções da inequação $\sum_{i=1}^{n} x_i \leq d$, onde d é um número inteiro positivo e todas as variáveis x_i são inteiras não negativas?

2. Dados n letras α e d letras β, quantas palavras diferentes podemos formar com todas estas letras permutando-as?

A princípio, estes dois problemas podem parecer distintos, mas são equivalentes. Podemos fazer a seguinte associação para os dois problemas. Dada uma permutação qualquer de n letras α e d letras β, o número de letras β à esquerda da primeira letra α corresponde ao valor da variável x_1. O número de letras β entre a i-ésima e $(i+1)$-ésima letras α corresponde ao valor da variável x_{i+1}, para $1 \leq i \leq n-1$. O número de letras β à direita da última letra α não é associado com nenhuma variável.

Para respondermos às duas questões acima usaremos combinações. A combinação $\binom{n}{d}$ representa o número total de subconjuntos de d elementos de um determinado conjunto com n elementos. O número destes subconjuntos é exatamente:

$$\frac{n!}{(n-d)!d!}.$$

Dados n letras α e d letras β, podemos enumerar as posições onde cada letra pode aparecer. As posições vão de 1 até $n+d$. O número de possibilidades de distribuição das letras α nestas posições é dado por $\binom{n+d}{n}$. Dado cada uma destas possibilidades, restam $n+d-n$ posições para serem preenchidas pelas letras β, o que nos dá $\binom{d}{d}$ possibilidades. Logo o número total de palavras diferentes que podemos formar com n letras α e d letras β é dado por

$$\binom{n+d}{n}\binom{d}{d} = \binom{n+d}{n} = \binom{n+d}{d}.$$

O número de configurações para os dois problemas propostos no início desta seção é dado pela fórmula acima. Nesta tese, usamos este resultado no contexto de empacotamento. Suponha que para uma determinada instância do problema *bin packing* unidimensional, o número total de itens seja m e saibamos calcular o número total de configurações diferentes de recipientes. Denotamos por c o número total de configurações de recipientes.O número total de soluções para esta instância é limitado por

$$\binom{m+c}{m}.$$

De fato, note que o número máximo de recipientes utilizados em uma solução qualquer para esta instância é m. Associando cada configuração i de recipiente a uma variável inteira x_i, estamos contando o número de possibilidades de soluções para uma equação

$$\sum_{i=1}^{c} x_i \leq m,$$

onde cada x_i indica quantas vezes a configuração i será usada em uma solução específica.

Logo o número de soluções para esta instância é limitado por

$$\binom{m+c}{m} = \frac{(m+c)!}{m!c!} \leq (m+c)^c.$$

Note que se o número de configurações de recipientes for constante, o número total de soluções para esta determinada instância é polinomial em relação ao tamanho da entrada.

2.4 Geração de Colunas

Nesta seção apresentamos de forma resumida o funcionamento do algoritmo *simplex*, e como podemos resolver programas lineares com um número muito grande de colunas utilizando-se o método de geração de colunas. Maiores detalhes sobre programação linear, o algoritmo *simplex* e geração de colunas podem ser encontrados no livro de Bazaraa *et al.* [6].

2.4.1 O algoritmo *Simplex*

Considere o programa linear,

$$\begin{array}{rll} \text{Min} & cx & \\ \text{sujeito a} & Ax = b & \\ & x_j \geq 0 \quad j = 1, \ldots, n. & \end{array} \tag{2.1}$$

onde A é uma matriz $m \times n$ de posto m, c é um vetor de custos de tamanho n, b é um vetor de tamanho m e x é um vetor de variáveis de tamanho n. Uma *base* $B_{m\times m}$ deste programa linear

consiste em uma sub-matriz de A inversível. Uma base é dita *viável* se $B^{-1}b \geq \mathbf{0}$ (note que $B^{-1}b$ é uma solução do programa linear).

A idéia do algoritmo *simplex* para resolução de sistemas deste tipo está fundamentada em alguns resultados que relacionam *bases viáveis* e pontos extremos do poliedro descrito pelas m restrições do sistema.

Teorema 2.4.1 *Se o programa linear (2.1) possui uma solução ótima com valor finito, então existe um vértice do poliedro descrito pelas restrições de (2.1), cujo valor é igual ao valor da solução ótima.*

Em outras palavras o que o teorema nos diz, é que é suficiente nos concentrarmos nos pontos extremos do poliedro pois se um programa linear possui uma solução ótima finita, então podemos achar uma solução ótima em algum vértice.

Seja B uma base viável para (2.1). Seja x_B as variáveis correspondentes as colunas de B (chamadas de *variáveis básicas*) e x_N as demais variáveis (chamadas *variáveis não básicas*) correspondentes as colunas de uma matriz N de dimensões $m \times (n-m)$. Fazendo $x_B = B^{-1}b$ e $x_N = 0$, temos uma solução para o programa linear. Para cada base B possível, associamos esta com a solução (chamada *solução básica*) dada por $x_B = B^{-1}b$ e $x_N = 0$. O próximo teorema garante que cada uma das *soluções básicas* corresponde a um vértice, e que para cada vértice do poliedro existe uma solução básica correspondente.

Teorema 2.4.2 *Para cada vértice do poliedro do programa linear (2.1), existe uma (não necessariamente única) base viável que corresponde a este vértice, e para cada base viável existe apenas um vértice correspondente a esta base.*

A idéia do algoritmo *simplex* é partir de uma base viável inicial e fazer alterações de colunas que levem a outras bases melhorando o valor da solução até um ponto em que possamos garantir que estamos em uma solução ótima, ou que o sistema é ilimitado (no caso em que a solução pode ser melhorada o quanto quisermos).

Suponha que temos uma solução básica $\begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$ cujo valor da função objetivo é

$$z_0 = c \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix} = (c_B, c_N) \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix} = c_B B^{-1} b \tag{2.2}$$

Temos que $b = Ax = Bx_B + Nx_N$, e multiplicando esta equação por B^{-1} obtemos

$$\begin{aligned} x_B &= B^{-1}b - B^{-1}Nx_N \\ &= B^{-1}b - \textstyle\sum_{j \in R} B^{-1}a_j x_j \\ &= b^* - \textstyle\sum_{j \in R} (y_j) x_j \end{aligned} \tag{2.3}$$

onde R é o conjunto dos índices das variáveis não básicas e $b^* = B^{-1}b$. O valor da função objetivo pode ser reescrito da seguinte forma:

$$\begin{array}{rcl} z = & c_B x_B + c_N x_N & \\ = & c_B(B^{-1}b - \sum_{j\in R} B^{-1}a_j x_j) + \sum_{j\in R} c_j x_j & \\ = & c_B B^{-1}b - c_B \sum_{j\in R} B^{-1}a_j x_j + \sum_{j\in R} c_j x_j & (2.4) \\ = & z_0 - \sum_{j\in R}(c_B B^{-1}a_j - c_j)x_j & \\ = & z_0 - \sum_{j\in R}(z_j - c_j)x_j & \end{array}$$

onde $z_j = c_B B^{-1}a_j$ para cada $j \in R$ e $z_0 = c_B B^{-1}b$.

Desta forma, utilizando estas transformações podemos reescrever o sistema (2.1) da seguinte forma:

$$\begin{array}{rll} \text{Min} & z = z_0 - \displaystyle\sum_{j\in R}(z_j - c_j)x_j & \\ \text{sujeito a} & \displaystyle\sum_{j\in R}(y_j)x_j + x_B = b^* & (2.5) \\ & x_j \geq 0 \quad j \in R \quad \text{e } x_B \geq 0. & \end{array}$$

Note que partimos de uma base viável com solução (x_B, x_N) e reescrevemos o programa linear de tal forma que a função objetivo tem um valor constante z_0 menos um termo em função das variáveis não básicas, que tem valor zero dado que $x_N = 0$. Mas note que se para alguma coluna a_j de A tivermos

$$c_B B^{-1}a_j - c_j = z_j - c_j > 0$$

podemos incrementar o valor da variável não básica x_j correspondente, e melhorarmos o valor da solução do programa linear. Se para todo j tivermos $z_j - c_j \leq 0$ então a solução básica corresponde a uma solução ótima. O valor $c_j - z_j$ é conhecido como custo reduzido da coluna j e o vetor $c_B B^{-1}$ corresponde ao vetor de solução dual do programa linear. Uma observação interessante é que o custo reduzido das variáveis básicas é igual a zero. Para ver isso, note que BB^{-1} é a identidade, e para uma coluna b_i de B, temos que $c_B B^{-1}b_i = c_i$.

Seja k o índice de uma variável não básica cujo valor $z_k - c_k$ seja positivo. Mantendo todas as demais variáveis não básicas iguais a zero o sistema (2.5) pode ser reduzido a

$$z = z_0 - (z_k - c_k)x_k \tag{2.6}$$

e

$$\begin{bmatrix} x_{B_1} \\ x_{B_2} \\ . \\ . \\ . \\ x_{B_r} \\ . \\ . \\ . \\ x_{B_m} \end{bmatrix} = \begin{bmatrix} b_1^* \\ b_2^* \\ . \\ . \\ . \\ b_r^* \\ . \\ . \\ . \\ b_m^* \end{bmatrix} - \begin{bmatrix} y_{1k} \\ y_{2k} \\ . \\ . \\ . \\ y_{rk} \\ . \\ . \\ . \\ y_{mk} \end{bmatrix} x_k \tag{2.7}$$

Note que se $y_{ik} \leq 0$, então x_{B_i} deve aumentar seu valor a medida que x_k cresce. Se $y_{ik} > 0$, então x_{B_i} deve diminuir de valor a medida que x_k cresce. Para mantermos as restrições de não-negatividade, x_k pode ser incrementada até o ponto em que a primeira variável básica assume valor igual a zero. Examinando a equação (2.7), podemos ver que a primeira variável básica a assumir valor zero a medida que x_k aumenta é aquela correspondente a menor fração b_i^*/y_{ik} para y_{ik} positivo. Podemos incrementar x_k até o valor

$$x_k = \frac{b_r^*}{y_{rk}} = \text{Min}_{1 \leq i \leq m} \left\{ \ \frac{b_i^*}{y_{ik}} : y_{ik} > 0 \ \right\}. \tag{2.8}$$

Note que se $y_{ik} \leq 0$ para todo i, então o sistema é ilimitado, pois a medida que incrementamos x_k, as variáveis básicas também são incrementadas e o valor da função objetivo decresce. Quando x_k é incrementada criamos uma nova solução básica onde a coluna a_k da matriz A toma lugar da coluna a_{B_r}. Temos então uma nova solução cujos valores das variáveis básicas são

$$x_{B_i} = b_i^* - \frac{y_{ik}}{y_{rk}} b_r^*, \text{ para } i = 1, 2, \ldots, m$$

$$x_k = \frac{b_r^*}{y_{rk}}$$

e todas as demais variáveis iguais a zero.

Abaixo temos uma descrição do algoritmo *simplex*. Maiores detalhes sobre o algoritmo, sobre como achar uma base viável inicial, tratamento de degenerescência, ciclagem e outros tópicos podem ser encontrados no livro de Bazaraa *et al.* [6].

Partindo de uma base B viável para o sistema (2.1) execute os seguintes passos.

1. Resolva o sistema $Bx_B = b$ cuja única solução é $x_B = B^{-1}b = b^*$. Seja $x_B = b^*$, $x_N = 0$ e $z = c_B x_B$. Prossiga para o próximo passo.

2. Resolva o sistema $wB = c_B$ cuja solução é $w = c_B B^{-1}$. Calcule $z_j - c_j = wa_j - c_j$ para todas as variáveis não básicas. Seja

$$z_k - c_k = \text{Max}_{j \in R} \{z_j - c_j\}$$

onde R é o conjunto dos índices associados as variáveis não básicas. Se $z_k - c_k \leq 0$ então pare com a solução básica atual como uma solução ótima. Caso contrário prossiga para o próximo passo.

3. Resolva o sistema $By_k = a_k$, cuja solução é $y_k = B^{-1}a_k$. Se $y_k \leq \mathbf{0}$ então pare pois solução é ilimitada. Se $y_k \not\leq \mathbf{0}$ prossiga para o próximo passo.

4. Calcule o índice r da coluna a sair da base que é aquele que atem ao mínimo

$$\frac{b_r^*}{y_{rk}} = \text{Min}_{1 \leq i \leq m} \left\{ \frac{b_i^*}{y_{ik}} : y_{ik} > 0 \right\}.$$

Atualize a matriz B inserindo a coluna a_k no lugar da coluna a_{B_r}. Volte ao passo 1.

2.4.2 O algoritmo *Simplex* com Geração de Colunas

A idéia do algoritmo de geração de colunas é simular o algoritmo *simplex*, mas no passo 2 do algoritmo, ao invés de calcularmos o custo $z_j - c_j$ para cada uma das variáveis não básicas, resolvemos de forma indireta o sub-problema

$$\text{Max}_{j \in R} \{z_j - c_j\}. \tag{2.9}$$

Logo o algoritmo não mantém em memória todas as colunas do programa linear, e na hora de gerar uma coluna resolvendo o sub-problema (2.9) não há uma verificação explícita de todas as variáveis não básicas. Ressaltamos que a resolução de (2.9) implicitamente não é sempre possível mas para alguns problemas isto é possível.

Como exemplo do método de geração de colunas vamos considerar o problema *cutting stock* unidimensional (denotado por CS). Neste problema temos uma lista de itens $L = (a_1, \ldots, a_m)$, com tamanhos $(s(a_1), \ldots, s(a_m))$, um vetor de demandas para cada item $(d_1, \ldots, d_m)$ e recipientes (*bins*) de tamanho B. O objetivo do problema é gerar um empacotamento dos itens suprindo todas as demandas utilizando a menor quantidade de recipientes possível.

Chamamos de *padrão*, uma descrição de um empacotamento de itens em um recipiente. Podemos considerar um padrão $p_j = (p_{1j}, \ldots, p_{mj})$ como um vetor onde cada posição p_{ij} indica quantos itens do tipo i estão empacotados neste padrão j. Um padrão p_j para ser válido deve satisfazer $\sum_{i=1}^{m} p_{ij} s(a_i) \leq B$. Seja P uma matriz de dimensões $m \times n$ que contém todos

os tipos de padrões possíveis como colunas. Note que o número de padrões possíveis é muito grande. Podemos formular o problema CS da seguinte forma

$$\begin{array}{rl} \text{Min} & (\mathbf{1}) \cdot x \\ \text{sujeito a} & Px = d \\ & x_j \geq 0 \quad j = 1, \ldots, n. \end{array} \tag{2.10}$$

Neste programa linear temos um vetor x de tamanho n, que indica quantas vezes cada padrão deve ser utilizado em uma solução e (**1**) representa um vetor de uns com dimensão n. Note que estamos minimizando o número de recipientes utilizados.

O custo de uma coluna j deste programa linear é

$$z_j - c_j = (\mathbf{1})B^{-1}p_j - 1.$$

Seja $w = (\mathbf{1})B^{-1}$ e $\mathcal{P}$ o conjunto de padrões possíveis. No passo 2 do algoritmo *simplex* temos então que resolver o sub-problema

$$\text{Max}_{p_j \in P} \left\{ wp_j - 1 \right\},$$

onde cada padrão possível deve satisfazer

$$\sum_{i=1}^{m} p_{ij} s(a_i) \leq B.$$

Note que se considerarmos p_{ij} como variáveis inteiras, o sub-problema acima corresponde ao problema da mochila, ou seja, ao invés de considerarmos todos os padrões possíveis, podemos resolver o sub-problema acima como um problema da mochila descrito abaixo

$$\begin{array}{rl} \text{Max} & wp - 1 \\ \text{sujeito a} & \displaystyle\sum_{i=1}^{m} p_i s(a_i) \leq B \\ & p_i \geq 0, \text{ inteira}, \quad i = 1, \ldots, m. \end{array} \tag{2.11}$$

onde p é um vetor de tamanho m de variáveis inteiras.

Podemos começar a resolução do programa linear (2.10) com uma base viável correspondente a matriz identidade $I_{m \times m}$, e o passo 2 do algoritmo *simplex* é resolvido utilizando o programa linear (2.11). Desta maneira, sempre mantemos na memória apenas as colunas básicas na resolução de (2.10), e o passo 2 do algoritmo *simplex* é resolvido de forma implícita sem a necessidade de calcular o custo $z_j - c_j$ para todas as variáveis não básicas.

Capítulo 3

Resumo dos Resultados

Neste capítulo descrevemos os principais resultados apresentados nesta tese. Cada um dos próximos capítulos desta tese representa um artigo com resultados originais desenvolvidos durante o doutorado. Cada artigo apresenta aplicações dos problemas considerados, bem como algoritmos propostos para tais problemas.

No Capítulo 4 apresentamos o problema que chamamos de *Class Constrained Shelf Bin Packing* (CCSBP). Este problema é uma generalização do *bin packing* onde itens têm classes diferentes e devemos empacotar os itens separando-os por prateleiras.

Uma instância para este problema consiste de uma tupla $I = (L, s, c, d, \Delta, B)$, onde L é uma lista de itens, s e c são funções de tamanho e classe sobre os itens de L, d é o tamanho de uma divisória, Δ é o tamanho máximo de uma prateleira e B é o tamanho dos recipientes. Dado uma sub-lista $L' \subseteq L$ denotamos por $s(L')$ a soma dos tamanhos dos itens em L', i.e, $s(L') = \sum_{e \in L'} s(e)$. Um empacotamento $\mathcal{P}$ da instância I para o problema CCSBP consiste em um conjunto de recipientes $\mathcal{P} = \{P_1, \ldots, P_k\}$, onde os itens em cada recipiente $P_i \in \mathcal{P}$ estão particionados em prateleiras $\{N_1^i, \ldots, N_{q_i}^i\}$ tal que para cada prateleira N_j^i temos que $s(N_j^i) \leq \Delta$, todos os itens em N_j^i são de uma mesma classe e $\sum_{j=1}^{q_i}(s(N_j^i) + d) \leq B$.

Uma aplicação interessante do problema CCSBP pode ser encontrada no trabalho de Ferreira *et al.* [16] que introduziram este problema que aparece na indústria de metais.

Apresentamos algoritmos baseados nas estratégias *First Fit (Decreasing)* e *Best Fit (Decreasing)* para o problema CCSBP. Quando o número de classes diferentes é limitado por uma constante, apresentamos algoritmos com fatores de aproximação assintótico 3.4 e 2.445. Se o número de classes não é limitado por uma constante, mostramos algoritmos com fatores de aproximação absolutos iguais a 4 e 3.

Por fim, para o caso em que o número de classes é limitado por uma constante, apresentamos um APTAS para o problema CCSBP.

Uma versão resumida deste artigo foi aceita para apresentação no GRACO2005 (2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics) [44]. A versão completa apresen-

tada nesta tese, foi aceita para publicação em uma edição especial da revista *Discrete Applied Mathematics* com artigos selecionados do congresso.

No Capítulo 5 consideramos dois problemas: o CCSBP e o problema *bin packing* com restrições de classes, denotado por CCBP. Uma instância deste último problema é uma tupla $I = (L, s, c, C, Q)$ onde $L = (a_1, \ldots, a_n)$ é uma lista com n itens, cada item $a_i \in L$ com tamanho $0 < s(a_i) \leq 1$ e classe $c(a_i) \in \{1, \ldots, Q\}$, e um conjunto de recipientes de tamanho 1 e C compartimentos. Um empacotamento para esta instância consiste em um conjunto de recipientes $\mathcal{P} = \{P_1, \ldots, P_k\}$ tal que, para cada P_i o número de classes diferentes de itens empacotados em P_i é no máximo C, e a soma dos tamanhos dos itens empacotados em P_i é no máximo 1. O objetivo do problema é encontrar um empacotamento de I que utiliza o menor número de recipientes possível.

Neste capítulo apresentamos esquemas de aproximação duais para ambos os problemas. O artigo que corresponde a este capítulo foi submetido para publicação em uma revista.

No Capítulo 6 apresentamos o problema *bin packing* com restrição de classes (CCBP) com aplicações para um problema de construção de servidores de vídeo sob demanda.

Para o problema *online*, consideramos dois casos: no primeiro caso, que chamamos de limitado, dada uma constante k, um algoritmo pode manter ativo no máximo k recipientes durante sua execução (conhecido na literatura como k-*bounded*); no segundo caso um número ilimitado de recipientes pode permanecer ativo. Um recipiente ativo é aquele que pode ser usado para empacotar itens. Quando um recipiente se torna inativo, ele não pode mais voltar a ser ativo. Para o caso limitado, mostramos que não pode existir nenhum algoritmo com fator de aproximação constante. Além disso, mostramos que se os itens de uma instância têm tamanho pelo menos ε, então não existe algoritmo com fator de aproximação melhor do que $\mathrm{O}(1/C\varepsilon)$. Para o caso não limitado mostramos um algoritmo *online* com fator de aproximação entre 2.666 e 2.75.

Também apresentamos neste capítulo resultados para a versão *offline* do problema. Quando todos os itens têm tamanhos iguais, apresentamos um algoritmo $(1 + 1/C)$-aproximado. Para o caso paramétrico, quando os itens possuem tamanhos no máximo B/m (B é o tamanho do recipiente), para algum inteiro m, apresentamos um algoritmo com fator de aproximação igual a $(1 + 1/C + 1/\min\{C, m\})$. Implementamos alguns dos algoritmos apresentados e reportamos resultados computacionais baseados em instâncias que refletem o problema de construção de servidores de vídeo sob demanda. Tais experimentos mostram que os algoritmos considerados geram soluções de muito boa qualidade.

Neste capítulo consideramos ainda a versão do problema com recipientes de tamanhos variados (VCCBP). Este problema foi estudado primeiramente por Dawande *et al.* [12, 11] onde uma tentativa de um APTAS foi considerada, para o caso em que o número de classes diferentes na entrada é limitado por uma constante. Mostramos que o algoritmo proposto por Dawande *et al.* [12, 11] está errado e então mostramos um APTAS para o problema.

Os resultados deste capítulo foram apresentados no *12th Annual International Computing and Combinatorics Conference (COCOON 2006)* [45], e a versão apresentada aqui foi submetida para publicação em uma revista.

No Capítulo 7 apresentamos algoritmos de aproximação para a versão do problema *bin packing* onde os itens possuem demandas, ou seja, para cada item existe uma multiplicidade que indica quantos itens deste tamanho devem ser empacotados. Estes problemas são conhecidos na literatura como problemas de *cutting stock*. Neste capítulo mostramos como adaptar vários algoritmos de aproximação desenvolvidos para problemas sem demanda para o caso onde há demanda. Mostramos que se um determinado algoritmo para um problema sem demanda tiver uma determinada propriedade, que denominamos de algoritmos comportados, então este algoritmo pode ser transformado em outro para o caso com multiplicidades. Dentre os resultados deste capítulo destacamos um esquema de aproximação assintótico para o problema *cutting stock* unidimensional e um algoritmo com fator de aproximação assintótico igual 2.077 para o problema *cutting stock* bidimensional. Os resultados deste capítulo aparecem em um artigo aceito para publicação na revista *European Journal of Operational Research* [8].

Finalmente no Capítulo 8 apresentamos algoritmos para problemas de empacotamento bidimensional. Os problemas considerados assumem cortes guilhotináveis e em estágios. Apresentamos algoritmos exatos para o problema da mochila bidimensional baseados em programação dinâmica. Consideramos também o problema *bin packing* bidimensional com demandas e o problema *strip packing* bidimensional com demandas. Para estes problemas apresentamos heurísticas baseadas no método de geração de colunas. Implementamos os algoritmos propostos e reportamos os resultados computacionais obtidos com estes algoritmos. Tais resultados indicam que estes algoritmos acham soluções de muito boa qualidade em tempos razoáveis. Os resultados deste capítulo, juntamente com resultados obtidos anteriormente por Cintra e Wakabayashi [9], fazem parte de um artigo aceito para publicação na revista *European Journal of Operational Research*.

Capítulo 4

Artigo: *A One-Dimensional Bin Packing Problem with Shelf Divisions*

E. C. Xavier[2] F. K. Miyazawa[2]

Abstract

Given bins of size B, non-negative values d and Δ, and a list L of items, each item $e \in L$ with size s_e and class c_e, we define a shelf as a subset of items packed inside a bin with total items size at most Δ such that all items in this shelf have the same class. Two subsequent shelves must be separated by a shelf division of size d. The size of a shelf is the total size of its items plus the size of the shelf division. The Class Constrained Shelf Bin Packing Problem (CCSBP) is to pack the items of L into the minimum number of bins, such that the items are divided into shelves and the total size of the shelves in a bin is at most B. We present hybrid algorithms based on the First Fit (Decreasing) and Best Fit (Decreasing) algorithms, and an APTAS for the problem CCSBP when the number of different classes is bounded by a constant C.

Key Words: Approximation algorithms, bin packing, shelf packing.

4.1 Introduction

In this paper we present approximation algorithms for a class constrained bin packing problem when the items must be separated by non-null shelf divisions. We denote this problem by *Class Constrained Shelf Bin Packing Problem* (CCSBP).

[1]An extended abstract of this paper was presented at GRACO2005 (2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics) and appeared in Electronic Notes in Discrete Mathematics 19 (2005) 329–335. This research was partially supported by CNPq (Proc. 470608/01–3, 478818/03–3, 306526/04-2 and 490333/04-4) and ProNEx–FAPESP/CNPq (Proc. 2003/09925-5).

[2]Instituto de Computação — Universidade Estadual de Campinas, Caixa Postal 6176 — 13084–971 — Campinas–SP — Brazil, {eduardo.xavier,fkm}@ic.unicamp.br.

An instance for the CCSBP problem is a tuple $I = (L, s, c, d, \Delta, B)$, where L is a list of items, s and c are size and class functions over L, d is the size of the shelf division, Δ is the maximum size of a shelf and B is the size of the bins. Given a sublist of items $L' \subseteq L$ we denote by $s(L')$ the sum of the sizes of the items in L', i.e, $s(L') = \sum_{e \in L'} s_e$. A *shelf packing* $\mathcal{P}$ of an instance I for the CCSBP problem is a set of bins $\mathcal{P} = \{P_1, \ldots, P_k\}$, where the items packed in a bin $P_i \in \mathcal{P}$ are partitioned into shelves $\{N_1^i, \ldots, N_{q_i}^i\}$ such that for each shelf N_j^i we have that $s(N_j^i) \leq \Delta$, all items in N_j^i are of the same class and $\sum_{j=1}^{q_i}(s(N_j^i) + d) \leq B$. Without loss of generality we consider that $0 < s_e \leq \Delta$ and $c_e \in \mathbb{Z}^+$ for each $e \in L$.

The CCSBP problem is to find a shelf packing of the items of L into the minimum number of bins. This problem is NP-hard since it is a generalization of the bin packing problem: in this case consider that the instance has just one class, $\Delta = B$ and $d = 0$. We note that the term shelf is used under another context in the literature for the 2-D strip packing problem. In this case, packings are two staged packings divided into levels.

There are many practical applications for the CCSBP problem even when there is only one class of items. For example, when the items to be packed must be separated by non-null shelf divisions (inside a bin) and each shelf has a limited capacity. In Figure 4.1 we can see an example of a shelf packing of items into one bin, with $B = 60$, $\Delta = 17$, $d = 3$ and all items of the same class. The CCSBP problem is also adequate when some items cannot be stored in a same shelf (like foods and chemical products). In most of the cases, the sizes of the shelf divisions have non-negligible width. Although these problems are very common in practice, to our knowledge this is the first paper that presents approximation results for them.

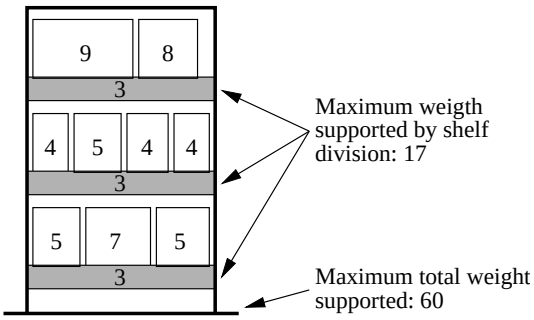

Figure 4.1: Example of a shelf packing of items into one bin.

An interesting application for the CCSBP problem was introduced by Ferreira et al. [4] in the iron and steel industry. In this problem, we have raw material rolls that must be cut into final rolls grouped by certain properties after two cutting phases. The rolls obtained after the first phase, called primary rolls, are submitted to different processing operations (tensioning, tempering, laminating, hardening etc.) before the second phase cut. Due to technological limitations, primary rolls have a maximum allowable width and each cut has a trimming process

that generates a loss in the roll width. Each processing operation has a high cost which implies items to be grouped before doing it, where each group corresponds to one shelf.

Given an algorithm $\mathcal{A}$, and an instance I for the CCSBP problem, we denote by $\mathcal{A}(I)$ the number of bins used by algorithm $\mathcal{A}$ to pack the instance I and by $\mathrm{OPT}(I)$ the number of bins used in an optimal solution. The algorithm $\mathcal{A}$ is an α-approximation, if $\mathcal{A}(I)/\mathrm{OPT}(I) \leq \alpha$, for any instance I. In this case, we also say that $\mathcal{A}$ has an absolute performance bound α. In bin packing problems, it is also usual to use the asymptotic worst case analysis. We say that $\mathcal{A}$ has an asymptotic performance bound α if there is a constant β such that $\mathcal{A}(I) \leq \alpha\mathrm{OPT}(I) + \beta$ for any instance I.

Given an algorithm A_ε, for some $\varepsilon > 0$, and an instance I for some problem P we denote by $A_\varepsilon(I)$ the value of the solution returned by algorithm A_ε when executed on instance I. We say that A_ε, for $\varepsilon > 0$, is an asymptotic polynomial time approximation scheme (APTAS) for the problem CCSBP if there exist constants t and K such that $A_\varepsilon(I) \leq (1 + t\varepsilon)\mathrm{OPT}(I) + K$ for any instance I.

Results: In this paper we present hybrid algorithms for the CCSBP problem, based on the First Fit (Decreasing) and Best Fit (Decreasing) algorithms for the bin packing problem. When the number of different classes is bounded by a constant, we show that the hybrid versions of the First Fit and Best Fit algorithms have an asymptotic performance bound of 3.4 and the hybrid versions of the First Fit Decreasing and Best Fit Decreasing algorithms have an asymptotic performance bound less than 2.445. In the case where the number of different classes is part of the input, we show that the hybrid versions of the First Fit and Best Fit algorithms have an absolute performance bound of 4 and the hybrid version of the First Fit Decreasing algorithm has an absolute performance bound of 3. At last, for the case when the number of classes is bounded by a constant, we present an APTAS for the CCSBP problem.

Related Work: A special case of the CCSBP problem is the Bin Packing problem, which is one of the most studied problems in the literature. Some of the most famous algorithms for the bin packing problem are the algorithms FF, BF, FFD and BFD, with asymptotic performance bounds $17/10$, $17/10$, $11/9$ and $11/9$, respectively. Fernandez de la Vega and Lueker [3] presented an APTAS for the bin packing problem. Dawande et al. [2], presented approximation schemes for a class constrained version of the bin packing (CCBP), where bins can have different sizes and each bin is used to pack items of at most k different classes, and the number of different classes in the input instance is bounded by a constant. Shachnai and Tamir [7], presented a polynomial time approximation scheme for a dual version of the problem CCBP also for the case where the number of different classes in the input instance is bounded by a constant. Shachnai and Tamir [8], considered a special case of an online class constrained bin packing problem. In this case all items have the same size and must be packed without knowledge of the next subsequent items of the input. We refer the reader to Coffman et al. [1] for a survey on approximation algorithms for bin packing problems. In [11], we consider the knapsack version

of the CCSBP problem, where each item e has also a value v_e. The objective is to find a shelf packing of a subset S of the items in just one knapsack (bin) of size K, such that the total value of the items in S is maximum. We also give a PTAS for this problem. We remark that, despite of the similarity of the problems, the techniques and algorithms used in this paper are not related to the ones used in the knapsack version of the problem. Practical approaches for the CCSBP problem were considered by Ferreira et al. [4], that introduced the problem in the iron and steel industry. Recently, the problem was considered by Hoto et al. [5] and Marques and Arenales [6]. Hoto et al. [5], considered the cutting stock version of the problem where a demand of items must be attended by the minimum number of bins. They use a column generation strategy. In [6] exact and heuristic algorithms are presented for a knapsack version of the problem.

4.1.1 Notation

Given an instance $I = (L, s, c, d, \Delta, B)$ for the CCSBP problem, we denote by $n = |L|$ the number of items in this instance. For any integer t, we denote by $[t]$ the set $\{1, \ldots, t\}$. We assume that each class belongs to the set $[C]$. We assume that C is bounded by a constant, unless otherwise stated. We denote by $\mathrm{OPT}_s(I)$ the minimum number of non-null shelves in an optimal packing of I, and by $\mathrm{OPT}(I)$ the number of bins in this optimal solution. Given a packing $\mathcal{P} = \{P_1, \ldots, P_k\}$, we denote by $|\mathcal{P}| = k$ the number of bins used in this packing, and by $N_s(\mathcal{P})$ the number of shelves used in all bins of $\mathcal{P}$. Given an algorithm $\mathcal{A}$ we denote by $\mathcal{A}(I)$ the number of bins used by the algorithm $\mathcal{A}$ to pack the instance I.

4.1.2 Simple Lower Bounds

The following facts present lower bounds for the number of bins used in any optimum solution for the CCSBP problem.

Fact 4.1.1 *For any instance $I = (L, s, c, d, \Delta, B)$, we have*

$$\mathrm{OPT}(I) \geq \frac{s(L)}{\lceil B/(d+\Delta)\rceil \Delta}.$$

Proof. Since $\lceil B/(d+\Delta)\rceil$ is an upper bound for the number of totally filled shelves in a bin, the total items size in a bin is at most $\lceil B/(d+\Delta)\rceil\Delta$. □

Fact 4.1.2 *For any instance $I = (L, s, c, d, \Delta, B)$, we have*

$$\mathrm{OPT}(I) \geq \frac{s(L) + \mathrm{OPT}_s(I)d}{B} \geq \frac{s(L) + \lceil s(L)/\Delta\rceil d}{B}.$$

Proof. The statement holds since $\lceil s(L)/\Delta\rceil$ is a lower bound for the number of shelves used in any packing. □

4.2 Hybrid versions of the First Fit and Best Fit Algorithms

In this section we present hybrid versions of the First Fit (Decreasing) and Best Fit (Decreasing) algorithms, for the classic bin packing problem, to the CCSBP problem. Without loss of generality, we assume that all bins have capacity 1.

We briefly describe how these algorithms work for the classic bin packing problem. The First Fit (FF) and the Best Fit (BF) algorithms pack the items of a given list $L = (e_1, \ldots, e_m)$ in the order given by L. Assume that the items $e_1, \ldots, e_{i-1}$ have been packed into the bins $B_1, B_2, \ldots, B_k$, each bin with capacity 1. To pack the next item e_i, the algorithm FF (resp. BF) finds the smallest index j, $1 \leq j \leq k$, such that $s(B_j) + s(e_i) \leq 1$ (resp. $s(B_j)$ is maximum given that $s(B_j) + s(e_i) \leq 1$). If the algorithm FF (resp. BF) finds such a bin, the item e_i is packed into the bin B_j. Otherwise, the item e_i is packed into a new bin B_{k+1}. This process is repeated until all the items of L have been packed.

The First Fit Decreasing (FFD) (resp. Best Fit Decreasing (BFD)) algorithm first sorts the items of L in non-increasing order of size and then apply the algorithm FF (resp. BF). The following result holds (see [1, 9]).

Theorem 4.2.1 *For any instance I for the bin packing problem, we have*

$$\text{FF}(I) \leq \frac{17}{10}\,\text{OPT}(I) + 1, \quad \text{BF}(I) \leq \frac{17}{10}\,\text{OPT}(I) + 1,$$

$$\text{FFD}(I) \leq \frac{11}{9}\,\text{OPT}(I) + 3, \quad \text{BFD}(I) \leq \frac{11}{9}\,\text{OPT}(I) + 3$$

$$\textit{and} \quad \text{FFD}(I) \leq \frac{3}{2}\text{OPT}(I).$$

Now we can present the hybrid algorithms for the problem CCSBP.

Algorithms SFF**,** SBF**,** SFFD **and** SBFD**:** Given an instance $I = (L, s, c, d, \Delta, B)$, the algorithm SFF (resp. SBF, SFFD and SBFD) uses the algorithm FF (resp. BF, FFD and BFD) to pack all items of a same class into shelves of size Δ. The algorithm considers the size of each generated shelf as the total items size in the shelf plus the size of the shelf division d. The set of generated shelves are then packed into bins of size B using the algorithm FF (resp. BF, FFD and BFD).

Given an instance $I = (L, s, c, d, \Delta, B)$ for the CCSBP problem, we denote by $\text{OPT}_\Delta(I)$ the minimum number of shelves of size Δ needed to pack L, where all items in a shelf have the same class. Clearly $\text{OPT}_\Delta(I)$ is a lower bound for the number of shelves used in any optimal solution. That is, $\text{OPT}_\Delta(I) \leq \text{OPT}_s(I)$.

Theorem 4.2.2 *Let I be an instance for the* CCSBP *problem. If the number of classes in I is bounded by C then*

$$\text{SFF}(I) \leq (3 + \frac{2}{5})\,\text{OPT}(I) + 2C, \quad \text{SBF}(I) \leq (3 + \frac{2}{5})\,\text{OPT}(I) + 2C,$$

$$\text{SFFD}(I) \leq (2 + \frac{4}{9})\,\text{OPT}(I) + 6C \quad \textit{and} \quad \text{SBFD}(I) \leq (2 + \frac{4}{9})\,\text{OPT}(I) + 6C.$$

Proof. Let $\mathcal{A}'$ be an algorithm in $\{\text{FF}, \text{BF}, \text{FFD}, \text{BFD}\}$ such that

$$\mathcal{A}'(I') \leq \alpha \text{OPT}(I') + \beta$$

for any instance I' of the classic bin packing problem and let $\mathcal{A}$ be the corresponding algorithm in $\{\text{SFF}, \text{SBF}, \text{SFFD}, \text{SBFD}\}$. Let $I = (L, s, c, d, \Delta, B)$ be an instance for the CCSBP problem. Consider the packing $\mathcal{P}$ produced by the algorithm $\mathcal{A}$ for the instance I. We consider that $|\mathcal{P}| > 1$, otherwise $\mathcal{P}$ is optimum. On average, all bins in $\mathcal{P}$ are filled by at least $1/2$ (including shelf divisions), since the algorithms pack the shelves in such a way that any pair of bins have total contents size greater than 1. We can conclude the following:

$$\begin{aligned}
\mathcal{A}(I)(1/2) &\leq s(L) + N_s(\mathcal{P})d \\
&\leq s(L) + (\alpha \text{OPT}_\Delta(I) + C\beta)d && (4.1)\\
&\leq s(L) + (\alpha \text{OPT}_s(I) + C\beta)d && (4.2)\\
&\leq \alpha(s(L) + d\text{OPT}_s(I)) + C\beta d \\
&\leq \alpha \text{OPT}(I) + C\beta, && (4.3)
\end{aligned}$$

where (4.1) holds from Theorem 4.2.1, and (4.3) follows from Fact 4.1.2 and the fact that $d \leq 1$. □

Notice that any algorithm for the classic bin packing problem can be easily extended to an algorithm with the same asymptotic performance bound and that produces bins that on average are filled by at least half of its capacities. Therefore, the following result can be easily derived as a generalization of the previous theorem.

Corollary 4.2.3 *Given an algorithm $\mathcal{A}'$ for the bin packing problem, such that*

$$\mathcal{A}'(L) \leq \alpha \text{OPT}(L) + \beta$$

for any instance L, then there exists an algorithm $\mathcal{A}$ for the CCSBP *such that*

$$\mathcal{A}(I) \leq 2\alpha \text{OPT}(I) + 2\beta C,$$

for any instance I of the CCSBP *problem.*

This result shows that when the number of classes is bounded by a constant we can obtain, using an APTAS for the bin packing problem, algorithms for the CCSBP problem with asymptotic performance bound as close to 2 as desired (although with high time complexity and with high value of β).

Now we consider that the number of different classes is not bounded by a constant. Notice that if a given algorithm $\mathcal{A}'$ for the bin packing problem has absolute performance bound α, then we can derive an algorithm $\mathcal{A}$ for the CCSBP problem with absolute performance bound 2α, even if the number of different classes of items is given as part of the input. Using the fact that algorithms FF, BF and FFD have absolute performance bound 2, 2 and $3/2$ respectively, we can obtain the following result.

Corollary 4.2.4 *Let I be an instance for the* CCSBP *problem, then*

$$\text{SFF}(I) \leq 4\text{OPT}(I), \quad \text{SBF}(I) \leq 4\text{OPT}(I) \quad and \quad \text{SFFD}(I) \leq 3\text{OPT}(I),$$

even if the number of different classes is not bounded by a constant.

From the practical point of view, the size of the shelf division d is not so large compared with Δ. The next theorem shows that if d is a small fraction of Δ, we can obtain a better performance bound for the Best and First Fit strategies.

Theorem 4.2.5 *Let $I = (L, s, c, d, \Delta, B)$ be an instance for the* CCSBP *problem. If the number of classes in I is bounded by C and $d = \frac{\Delta}{r}$, $r \geq 1$, we have*

$$\text{SFF}(I) \leq (2 + \frac{14}{5r})\,\text{OPT}(I) + 2C, \qquad \text{SBF}(I) \leq (2 + \frac{14}{5r})\,\text{OPT}(I) + 2C,$$

$$\text{SFFD}(I) \leq (2 + \frac{8}{9r})\,\text{OPT}(I) + 6C, \qquad \text{SBFD}(I) \leq (2 + \frac{8}{9r})\,\text{OPT}(I) + 6C,$$

$$and \quad \text{SFFD}(I) \leq (2 + \frac{2}{r})\text{OPT}(I).$$

Proof. Let $\mathcal{A}'$ be an algorithm in $\{\text{FF}, \text{BF}, \text{FFD}, \text{BFD}\}$ such that

$$\mathcal{A}'(I') \leq \alpha\text{OPT}(I') + \beta$$

for any instance I' of the classic bin packing problem and let $\mathcal{A}$ the corresponding algorithm in $\{\text{SFF}, \text{SBF}, \text{SFFD}, \text{SBFD}\}$. Let $I = (L, s, c, d, \Delta, B)$ be an instance for the CCSBP problem and $\mathcal{P}$ the packing produced by the algorithm $\mathcal{A}$ for the instance I.

We divide the proof in two cases, according to the values of $N_s(\mathcal{P})$ and $\text{OPT}_s(I)$.
CASE 1: $N_s(\mathcal{P}) < \text{OPT}_s(I)$. In this case, we have

$$\begin{aligned} \mathcal{A}(I)(1/2) &\leq s(L) + N_s(\mathcal{P})d \\ &< s(L) + \text{OPT}_s(I)d \\ &\leq \text{OPT}(I), \end{aligned} \tag{4.4}$$

where inequality (4.4) holds from Fact 4.1.2. That is,

$$\mathcal{A}(I) \leq 2\mathrm{OPT}(I). \tag{4.5}$$

CASE 2: $N_s(\mathcal{P}) \geq \mathrm{OPT}_s(I)$. In this case, we can follow the proof of Theorem 4.2.2 and obtain inequality (4.2). That is,

$$\mathcal{A}(I)(1/2) \leq s(L) + (\alpha\mathrm{OPT}_s(I) + C\beta)d. \tag{4.6}$$

Since on average each shelf generated by the algorithm $\mathcal{A}$ is filled by at least $\Delta/2$ (not including the shelf division), we have

$$\begin{aligned} \mathrm{OPT}(I) &\geq s(L) \\ &\geq N_s(\mathcal{P})(\Delta/2) \\ &\geq \mathrm{OPT}_s(I)\Delta/2 = \mathrm{OPT}_s(I)dr/2. \end{aligned}$$

That is, $\mathrm{OPT}_s(I)d \leq (2\mathrm{OPT}(I))/r$. Therefore, from inequality (4.6), we have

$$\begin{aligned} \mathcal{A}(I)(1/2) &\leq s(L) + (\alpha\mathrm{OPT}_s(I) + C\beta)d. \\ &= s(L) + \mathrm{OPT}_s(I)d + (\alpha-1)\mathrm{OPT}_s(I)d + C\beta d. \\ &\leq \mathrm{OPT}(I) + \frac{\alpha-1}{r}(2\mathrm{OPT}(I)) + C\beta d. \\ &\leq (1 + \frac{2(\alpha-1)}{r})\mathrm{OPT}(I) + C\beta d. \end{aligned}$$

That is,

$$\mathcal{A}(I) \leq (2 + \frac{4(\alpha-1)}{r})\mathrm{OPT}(I) + 2\beta C. \tag{4.7}$$

The theorem follows from inequalities (4.5), (4.7) and theorem 4.2.1. □

The following proposition shows that the previous theorem presents an asymptotic performance bound that is tight for the algorithms SFF and SBF, when d is very small compared to Δ.

Proposition 4.2.1 *The asymptotic performance bound of the algorithms* SFF *and* SBF *is at least 2, even when there is only one class.*

Proof. Let $I_n = (L, s, c, d, \Delta, B)$ be an instance with $L = (e_1, \ldots, e_{2n})$, $\varepsilon = 1/n$, $d = \varepsilon/2$, $B = 1$, $\Delta = 1/2$ and $s(e_i) = 1/2 - \varepsilon$ when i is odd and $s(e_i) = \varepsilon$ otherwise. Notice that $d = \Delta/n$. Also assume that all items have a same class. The SFF and SBF algorithms applied over this instance generates n shelves, each one containing one item of size $1/2 - \varepsilon$ and one item of size ε. The final packing generated by these algorithms has n bins, each one containing one shelf. An optimal packing with $n/2 + 1$ bins can be obtained in such a way that $n/2$ bins have two shelves each, one shelf with an item of size $1/2 - \varepsilon$, and the other shelf with two items, one item of size $1/2 - \varepsilon$ and another of size ε. The last bin contains the remaining items of size ε. □

4.3 An Asymptotic Polynomial Time Approximation Scheme

In this section we present an APTAS for the CCSBP problem when the number of different classes is bounded by a constant C.

The algorithm is presented in Figure 4.2 and is denoted by $\mathrm{ASBP}_\varepsilon$. It considers two cases: When $\varepsilon \geq d + \Delta$, it uses an algorithm denoted by $\mathrm{ASBP}'_\varepsilon$ and in the other case, it uses an algorithm denoted by $\mathrm{ASBP}''_\varepsilon$. Notice that the algorithm $\mathrm{ASBP}''_\varepsilon$ receives as input a rescaled instance so that the maximum shelf capacity is 1.

ALGORITHM $\mathrm{ASBP}_\varepsilon(L, s, c, d, \Delta, B)$

Input: List of items L, each item $e \in L$ with size s_e and class c_e, maximum capacity of a shelf Δ, shelf divisions of size d, bins of capacity $B = 1$.

Output: Shelf packing $\mathcal{P}$ of L.

Subroutines: Algorithms $\mathrm{ASBP}'_\varepsilon$ and $\mathrm{ASBP}''_\varepsilon$.

1. If $\varepsilon \geq d + \Delta$ then
2. $\quad \mathcal{P} \leftarrow \mathrm{ASBP}'_\varepsilon(L, s, c, d, \Delta, B)$
3. else
4. $\quad$ Scale the sizes d, Δ, B and s_e, for each $e \in L$, proportionally so that $\Delta = 1$.
5. $\quad$ // The condition to enter in this case is now equivalent to $\varepsilon \leq (d + \Delta)/B$.
6. $\quad \mathcal{P} \leftarrow \mathrm{ASBP}''_\varepsilon(L, s, c, d, \Delta, B)$.
7. Return $\mathcal{P}$.

Figure 4.2: Algorithm $\mathrm{ASBP}_\varepsilon$.

The intuition to consider these two cases is that in the first case, we can pack shelves almost optimally because the maximum size of a shelf is bounded by ϵ, and then the bins can be filled by at least $(1 - \epsilon)$. In the second case, since $\epsilon < d + \Delta$, we can bound by a constant the number of shelves used in each bin of an optimal solution. Then an enumeration step can be done to guess the shelves that are used in an optimal solution and an almost optimal shelf packing can be generated for large items. Small items are packed later using a linear programming strategy. In the following two subsections we show that algorithms $\mathrm{ASBP}'_\varepsilon$ and $\mathrm{ASBP}''_\varepsilon$ are APTAS.

4.3.1 The algorithm $\mathrm{ASBP}'_\varepsilon$

In this section we show that the algorithm $\mathrm{ASBP}'_\varepsilon$ is an APTAS for its corresponding case. This algorithm uses two subroutines: One is the FF algorithm and the other is an APTAS for the one dimensional bin packing problem presented by Fernandez de la Vega and Lueker [3]. We consider the version of this APTAS presented by Vazirani [10], which we denote by FL_ε, for which the following statement holds.

Theorem 4.3.1 *For any $\varepsilon > 0$, there exists a polynomial time algorithm FL_ε to pack a list of items L, each item $e \in L$ with size $s_e \in [0, \Delta]$, into bins of capacity Δ such that $\mathrm{FL}_\varepsilon(L) \leq (1+\varepsilon)\,\mathrm{OPT}_\Delta(L) + 1$, where $\mathrm{OPT}_\Delta(L)$ is the minimum number of bins of capacity Δ to pack L.*

The algorithm $\mathrm{ASBP}'_\varepsilon$ is presented in Figure 4.3. Given an instance I, the algorithm $\mathrm{ASBP}'_\varepsilon$ first packs all items of the instance into bins of size Δ using the algorithm FL_ε. The algorithm $\mathrm{ASBP}'_\varepsilon$ considers each one of these bins of size Δ as a shelf, where the size of a shelf is its total items size plus the size d of a shelf division. The algorithm $\mathrm{ASBP}'_\varepsilon$ packs these shelves into bins of size 1 using the algorithm FF.

ALGORITHM $\mathrm{ASBP}'_\varepsilon(L, s, c, \Delta, d, B)$

Input: List of items L, each item $e \in L$ with size s_e and class c_e, maximum capacity of a shelf Δ, shelf divisions of size d, bins of capacity $B = 1$ and $\varepsilon \geq d + \Delta$.

Output: Shelf packing $\mathcal{P}$ of L.

Subroutines: Algorithms FL_ε and FF.

1. Let L_c be the set of items of class c in L.

2. For each class $c \in [C]$ let $\mathcal{P}^c_\Delta$ be the packing of L_c obtained by the algorithm FL_ε using bins of capacity Δ.

3. Let $\mathcal{P}_\Delta$ be the union of the packings $\mathcal{P}^c_\Delta$, for each $c \in [C]$.

4. Consider each bin $D \in \mathcal{P}_\Delta$ as a shelf with size $\sum_{e \in D} s_e + d$.

5. Let S be the set of shelves obtained from $\mathcal{P}_\Delta$.

6. Let $\mathcal{P}$ be the packing obtained with the algorithm FF to pack the shelves of S into unit bins.

7. Return $\mathcal{P}$.

Figure 4.3: Algorithm $\mathrm{ASBP}'_\varepsilon$ where $\varepsilon \geq d + \Delta$.

The following statement holds for the algorithm $\mathrm{ASBP}'_\varepsilon$.

Lemma 4.3.2 *The algorithm $\mathrm{ASBP}'_\varepsilon$, is an APTAS for the* CCSBP *problem when the given instance I is such that $B = 1$ and $\varepsilon \geq d + \Delta$.*

Proof. In step 2, the algorithm obtains a packing $\mathcal{P}^c_\Delta$ of items of class c in L (items in L_c) into bins of capacity Δ using the algorithm FL_ε. By Theorem 4.3.1, we have

$$|\mathcal{P}^c_\Delta| \leq (1+\varepsilon)\mathrm{OPT}_\Delta(L_c) + 1. \tag{4.8}$$

The algorithm then considers each bin in $\mathcal{P}_\Delta$ as a shelf and obtains a shelf packing $\mathcal{P}$ using the algorithm FF to pack these shelves into unit bins. Since $\varepsilon \geq d + \Delta$, all bins of $\mathcal{P}$, except

perhaps the last, must be filled by at least $1-\varepsilon$. So,

$$\begin{aligned}(\mathrm{ASBP}'_\varepsilon(L)-1)(1-\varepsilon) &\leq s(L)+d|\mathcal{P}_\Delta| \\ &\leq s(L)+d\sum_{c=1}^{C}((1+\varepsilon)\mathrm{OPT}_\Delta(L_c)+1) \\ &\leq (1+\varepsilon)(s(L)+d\sum_{c=1}^{C}\mathrm{OPT}_\Delta(L_c))+dC \\ &\leq (1+\varepsilon)(s(L)+d\mathrm{OPT}_s(I))+dC \qquad (4.9) \\ &\leq (1+\varepsilon)\mathrm{OPT}(I)+C\end{aligned}$$

where inequality (4.9) is valid from Fact 4.1.2. Also notice that $d<1$. Therefore, for any $0<\varepsilon<1/3$ we have

$$\begin{aligned}\mathrm{ASBP}'_\varepsilon(L) &\leq \frac{1+\varepsilon}{1-\varepsilon}\mathrm{OPT}(I)+\frac{C}{1-\varepsilon}+1 \\ &\leq (1+3\varepsilon)\mathrm{OPT}(I)+\frac{3C}{2}+1.\end{aligned}$$

Notice that the running time of the algorithm $\mathrm{ASBP}'_\varepsilon$ only depends on the running times of algorithms FL_ε and FF, and the value of C. Let $T_{\mathrm{FL}}(n,\epsilon)$ and $T_{\mathrm{FF}}(n,\epsilon)$ be the running times of algorithms FL_ε and FF respectively. The running time of algorithm $\mathrm{ASBP}'_\varepsilon$ is $O(C\,T_{\mathrm{FL}}(n,\epsilon)+T_{\mathrm{FF}}(n,\epsilon))$. Since the algorithms FL_ε and FF have polynomial time complexity in n for fixed ε, the complexity time of algorithm $\mathrm{ASBP}'_\varepsilon$ is also polynomial in n for fixed ε. □

4.3.2 The algorithm $\mathrm{ASBP}''_\varepsilon$

Now, assume that the algorithm $\mathrm{ASBP}_\varepsilon$ obtains a shelf packing with the algorithm $\mathrm{ASBP}''_\varepsilon$. Throughout this section, we consider that s_e, d, Δ and B is the rescaled instance, such that $\Delta=1$. Notice that, the equivalent condition to enter in this case is

$$\varepsilon<\frac{d+\Delta}{B}=\frac{d+1}{B}. \qquad (4.10)$$

Notice that the maximum number of shelves completely filled packed in a bin is at most $\left\lceil\frac{B}{d+\Delta}\right\rceil$ which from (4.10) is at most $\frac{1}{\varepsilon}+1$. Observe that if there is any bin with more than $\frac{2}{\varepsilon}+2$ shelves of a same class, it has at least two shelves of this class with total size at most Δ. In this case, these two shelves can be combined into only one shelf. Without loss of generality we consider that each bin, in a solution for the CCSBP problem, contains at most $\frac{2}{\varepsilon}+2$ shelves of a same class.

In Figure 4.4 we present the algorithm $\mathrm{ASBP}''_\varepsilon$. The algorithm first obtains a pair $(\mathcal{P}_1,\mathbb{P})$ where $\mathcal{P}_1\cup\mathcal{P}'$, for each $\mathcal{P}'\in\mathbb{P}$, is a packing of big items (items with size at least ε^2). This pair

is obtained by the subroutine $\mathrm{A_{LR}}$. For each packing $\mathcal{P}_1 \cup \mathcal{P}'$, $\mathcal{P}' \in \mathbb{P}$, the algorithm $\mathrm{ASBP}''_\varepsilon$ uses the subroutine SMALL to pack the items with size less than ε^2 into the packing $\mathcal{P}'$. At least one of the generated packings uses at most $(1 + O(\varepsilon))\mathrm{OPT}(I) + O(1)$ bins as will be shown latter. The algorithm returns the packing with the smallest number of bins.

ALGORITHM $\mathrm{ASBP}''_\varepsilon(L, s, c, d, \Delta, B)$

Input: List of items L, each item $e \in L$ with size s_e and class c_e, maximum capacity of a shelf $\Delta = 1$, shelf divisions of size d, bins of capacity B and $\varepsilon \leq (d + \Delta)/B$.

Output: Shelf packing $\mathcal{P}$ of L.

Subroutines: Algorithms $\mathrm{A_{LR}}$ and SMALL.

1. Let G be the set of items $e \in L$ with size $s_e \geq \varepsilon^2$ and S the set $L \setminus G$.
2. Let $(\mathcal{P}_1, \mathbb{P})$ be a pair obtained from the algorithm $\mathrm{A_{LR}}$ applied over the list G.
3. For each $\mathcal{Q} \in \mathbb{P}$ do
4. let $\hat{\mathcal{Q}}$ be the packing obtained using the algorithm SMALL to pack S into $\mathcal{Q}$.
5. Let $\mathcal{P}$ be a packing $\mathcal{P}_1 \cup \hat{\mathcal{Q}}$ where $\mathcal{Q} \in \mathbb{P}$ and $|\hat{\mathcal{Q}}|$ is minimum.
6. Return $\mathcal{P}$.

Figure 4.4: Algorithm $\mathrm{ASBP}''_\varepsilon$.

In the next subsections we present the subroutines used by the algorithm $\mathrm{ASBP}''_\varepsilon$. The first subroutine called $\mathrm{A_{LR}}$ is used to generate a set of packings of big items. On the next subsection we present an algorithm called SMALL used to pack small items (items with size smaller than ϵ^2) in the packings of the big items generated by the algorithm $\mathrm{A_{LR}}$. In the last subsection we present the analysis of the algorithm $\mathrm{ASBP}''_\varepsilon$.

Generating Packings for the Big Items

In this section, we present the algorithm $\mathrm{A_{LR}}$ used to pack items with size at least ϵ^2 of a given input instance I. This algorithm generates a set of packings such that at least one can be used to pack the small items, such that the resulting packing has size at most $(1+O(\varepsilon))\mathrm{OPT}(I)+O(1)$. This algorithm uses the linear rounding technique, presented by Fernandez de la Vega and Lueker [3], and considers only items with size at least ε^2. The algorithm $\mathrm{A_{LR}}$ returns a pair $(\mathcal{P}_1, \mathbb{P})$, where $\mathcal{P}_1$ is a packing for a list of very big items and $\mathbb{P}$ is a set of packings for the remaining items.

We use the following notation in the description of the linear rounding technique: Given two lists of items X and Y, let $X_1, \ldots, X_C$ and $Y_1, \ldots, Y_C$ be the partition of X and Y respectively in classes, where X_c and Y_c have only items of class c for each $c \in [C]$. We write $X \preceq Y$ if

there is an injection $f_c : X_c \to Y_c$ for each $c \in [C]$ such that $s(e) \leq s(f(e))$ for all $e \in X_c$. Given two lists L_1 and L_2 we denote by $L_1 \| L_2$ the concatenation of these lists.

The algorithm $\mathrm{A_{LR}}$ uses three subroutines: $\mathrm{A_{ALL}}$, SFF, and $\mathrm{A_R}$. The algorithm SFF was presented in Section 4.2. In what follows we present the algorithms $\mathrm{A_{ALL}}$ and $\mathrm{A_R}$.

Algorithm $\mathrm{A_{ALL}}$**:** This is an algorithm used as subroutine to generate all possible packings with at most $\frac{2}{\varepsilon} + 2$ shelves of a same class, when the size of each item is bounded from below by a constant and the number of distinct sizes in each class is upper bounded by a constant t. The algorithm may generate empty shelves (used latter to pack small items). The following lemma guarantees the existence of such an algorithm.

Lemma 4.3.3 *Given an instance $I = (L, s, c, d, \Delta, B)$, with $\Delta = 1$, where the number of distinct items sizes in each class is at most a constant t, the number of different classes is bounded by a constant C and each item $e \in L$ has size $s_e \geq \varepsilon^2$, then there exists a polynomial time algorithm that generates all possible shelf packings of L with at most $\frac{2}{\varepsilon} + 2$ shelves of a same class in each bin.*

Proof. The number of items in a shelf is bounded by $p = 1/\varepsilon^2$. Given a class, the number of different shelves for it is bounded by $r' = \binom{p+t+1}{p}$ and so, the number of different shelves is bounded by $r = Cr'$. Since the number of shelves in a bin is bounded by $q = C(\frac{2}{\varepsilon} + 2)$, the number of different bins is bounded by $u = \binom{q+r}{q}$. Notice that u is a (large) constant since all the values p, q, r and u depends only on ε, C and t which are constants.

Therefore, the number of all feasible packings is bounded by $\binom{n+u}{n}$, which is bounded by $(n+u)^u$, which in turn is polynomial in n. □

Notice that the complexity time of the algorithm $\mathrm{A_{ALL}}$ is $O(n^{O(2C/\varepsilon)^{O(1/\varepsilon^2)^t}})$.

Algorithm $\mathrm{A_R}$**:** Given two lists X and Y such that $X \preceq Y$ and a packing $\mathcal{P}_Y$ of Y, there exists an algorithm, which we denoted by $\mathrm{A_R}$ (Replace), with input $(\mathcal{P}_Y, X)$, that obtains a packing $\mathcal{P}_X$ for X such that $|\mathcal{P}_X| = |\mathcal{P}_Y|$ as the next lemma guarantees.

Lemma 4.3.4 *If X and Y are two lists with $X \preceq Y$, then $\mathrm{OPT}(X) \leq \mathrm{OPT}(Y)$. Moreover, if $\mathcal{P}_Y$ is a shelf packing of Y then there exists a polynomial time algorithm $\mathrm{A_R}$ that given $\mathcal{P}_Y$ obtains a shelf packing $\mathcal{P}_X$ of X such that $|\mathcal{P}_X| = |\mathcal{P}_Y|$.*

Proof. The algorithm $\mathrm{A_R}$ sorts the lists X_c and Y_c for each $c \in [C]$ in non-increasing order of items size and then replaces in this order, each item of Y_c in the packing $\mathcal{P}_Y$ by an item of X_c. The possible remaining items of Y_c are removed. □

For any instance X, denote by $\overline{X}$ the instance with precisely $|X|$ items with size equal to the size of the smallest item in X. Clearly, $\overline{X} \preceq X$.

The algorithm $\mathrm{A_{LR}}$ is presented in Figure 4.5. It consists in the following: Let $G_1, \ldots, G_C$ be the partition of the input list G into classes $1, \ldots, C$ and let $n_c = |G_c|$ for each class c.

The algorithm $\mathrm{A_{LR}}$ partition each list G_c into groups $G_c^1, G_c^2, \ldots, G_c^{k_c}$. Let $G^1 = \cup_{c=1}^{C} G_c^1$. The algorithm generates a packing $\mathcal{P}_1$ of G^1 using $O(\varepsilon)\mathrm{OPT}(I)+1$ bins and a set $\mathbb{P}$ with polynomial number of packings for the items in $G \setminus G^1$. The packing $\mathcal{P}_1$ is generated by the algorithm SFF and the set of packings $\mathbb{P}$ is generated using the algorithms $\mathrm{A_{ALL}}$ and $\mathrm{A_R}$.

ALGORITHM $\mathrm{A_{LR}}(G)$

Input: List G with n items, each item $e \in G$ with size $s_e \geq \varepsilon^2$; maximum capacity of a shelf $\Delta = 1$; shelf divisions of size d and bins of capacity B.

Output: A pair $(\mathcal{P}_1, \mathbb{P})$, where $\mathcal{P}_1$ is a packing and $\mathbb{P}$ is a set of packings, where $\mathcal{P}_1 \cup \mathcal{P}'$ is a packing of G for each $\mathcal{P}' \in \mathbb{P}$.

Subroutines: Algorithms $\mathrm{A_{ALL}}$, SFF and $\mathrm{A_R}$.

1. Partition G into lists G_c for each class $c = 1, \ldots, C$ and let $n_c = |G_c|$.

2. Partition each list G_c into $k_c \leq \lceil 1/\varepsilon^3 \rceil$ groups $G_c^1, G_c^2, \ldots, G_c^{k_c}$, such that

$$G_c^1 \succeq G_c^2 \succeq \cdots \succeq G_c^{k_c},$$
$$\text{where } |G_c^j| = q_c = \lfloor n_c \varepsilon^3 \rfloor \text{ for all } j = 1, \ldots, k_c - 1,$$
$$\text{and } |G_c^{k_c}| \leq q_c.$$

3. Let $G^1 = \cup_{c=1}^{C} G_c^1$.

4. Let $\mathcal{P}_1$ be a packing of G^1 obtained by the algorithm SFF.

5. Let $\mathbb{Q}$ be the set of all possible packings obtained with the algorithm $\mathrm{A_{ALL}}$ over the list $(\overline{G_1^1} \| \ldots \| \overline{G_1^{k_1-1}} \| \ldots \| \overline{G_C^1} \| \ldots \| \overline{G_C^{k_C-1}})$.

6. Let $\mathbb{P}$ be the set of packings obtained with the algorithm $\mathrm{A_R}$ over each pair $(\mathcal{Q}, G_1^2 \| \ldots \| G_1^{k_1} \| \ldots \| G_C^2 \| \ldots \| G_C^{k_C})$, where $\mathcal{Q} \in \mathbb{Q}$.

7. Return $(\mathcal{P}^1, \mathbb{P})$.

Figure 4.5: Algorithm to obtain packings for items with size at least ε^2.

Denote by T_{ALL}, T_{R} and T_{SFF} the time complexity of algorithms $\mathrm{A_{ALL}}$, $\mathrm{A_R}$, and SFF respectively. The time complexity of steps 1–3 of algorithm $\mathrm{A_{LR}}$ is bounded by $O(n \log n)$. The overall time complexity of algorithm $\mathrm{A_{LR}}$ is $O(n \log n + T_{\mathrm{SFF}} + T_{\mathrm{ALL}} + T_{\mathrm{ALL}} T_{\mathrm{R}})$. Since T_{ALL}, T_{R} and T_{SFF} have polynomial time complexity in n for fixed ε, the time complexity of algorithm $\mathrm{A_{LR}}$ is also polynomial in n for fixed ε.

The following statement holds for the packing $\mathcal{P}_1$.

Lemma 4.3.5 *The packing $\mathcal{P}_1$ for the items in G^1 is such that*

$$|\mathcal{P}_1| \leq 4\varepsilon\, \mathrm{OPT}(I) + 1.$$

Proof. First, consider the total items size packed in a bin T of some shelf packing. From Fact

4.1.1 any optimum solution must satisfy

$$\mathrm{OPT}(I) \geq \frac{s(L)}{\lceil B/(d+\Delta)\rceil\,\Delta} = \frac{s(L)}{\lceil B/(d+1)\rceil} \geq \frac{1}{2}\,\frac{s(L)}{B/(d+1)}. \tag{4.11}$$

Notice that $\sum_{c=1}^{C} n_c = n$. The algorithm SFF packs at least $\lfloor B/(d+\Delta)\rfloor$ shelves in each bin, each shelf with at least one item. This means that each bin has at least $\lfloor B/(d+\Delta)\rfloor$ items, except perhaps the last, each item with size at least ε^2 and at most 1. Since the group G_1 has at most $n\varepsilon^3$ items, the number of bins in the shelf packing $\mathcal{P}_1$ can be bounded as follows.

$$\begin{aligned} |\mathcal{P}_1| &\leq \left\lceil \frac{n\varepsilon^3}{\lfloor B/(d+\Delta)\rfloor} \right\rceil = \left\lceil \frac{n\varepsilon^3}{\lfloor B/(d+1)\rfloor} \right\rceil \\ &\leq 2\frac{n\varepsilon^3}{B/(d+1)} + 1 \leq 2\frac{\varepsilon\, s(L)}{B/(d+1)} + 1 \\ &\leq 4\varepsilon\,\mathrm{OPT}(I) + 1, \end{aligned} \tag{4.12}$$

where the inequality (4.12) is valid from (4.11). □

Packing the Small Items

In this section we present an algorithm to pack the small items. If we only consider the big items, at least one of the packings generated by the algorithm $\mathrm{A_{LR}}$ has basically the same configuration of an optimal packing. That is, one of the generated packings has approximately the same number of bins and approximately the same shelves (including empty shelves that are used only for small items) of an optimal packing. Therefore the algorithm can guess how the small items are packed into the shelves of this packing, leaving only a small fraction of small items to be packed in new extra bins. Notice that a first approach to deal with the CCSBP problem, would be to produce the packing of the big items and then try to pack small items greedily. In the classic bin packing problem this approach works, since after packing the small items in the bins, each bin is filled by at least $(1-\epsilon)$ of its capacity, except perhaps the last bin. In the CCSBP problem this strategy may not work, since after packing the small items, the packing could use more shelves. This way, each bin would not be filled with items by at least $(1-\epsilon)$ of its capacity, since each bin also contains shelf divisions. To pack small items in the shelves generated by the algorithm $\mathrm{A_{LR}}$ we use a linear programming strategy. This approach has an easier and clearer analysis leading to the APTAS.

The algorithm $\mathrm{ASBP}''_{\varepsilon}$ uses a subroutine denoted by SMALL to pack small items (size less than ε^2) into a given packing of big items. Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a shelf packing of a list of items L and assume that we have to pack a set S of small items, with size at most ε^2, into $\mathcal{P}$. The packing of the small items is obtained from a solution of a linear program. Let $N_1^{ic}, \ldots, N_{n_{ic}}^{ic}$ be the shelves of class c in the bin P_i of the packing $\mathcal{P}$. For each shelf N_j^{ic}, define a non-negative

variable x_j^{ic}. The variable x_j^{ic} indicates the total size of small items of class c that is to be packed in the shelf N_j^{ic}. Consider the following linear program denoted by LPS:

$$\max \sum_{i=1}^{k}\sum_{c=1}^{C}\sum_{j=1}^{n_{ic}} x_j^{ic}$$

$$s(N_j^{ic}) + x_j^{ic} \leq \Delta \qquad \forall\, i \in [k],\ c \in [C],\ j \in [n_{ic}], \quad (1)$$

$$\sum_{c=1}^{C}\sum_{j=1}^{n_{ic}}(s(N_j^{ic}) + x_j^{ic} + d) \leq B \qquad \forall\, i \in [k], \quad (2)$$

$$\sum_{i=1}^{t}\sum_{j=1}^{n_{ic}} x_j^{ic} \leq s(S_c) \qquad \forall\, c \in [C], \quad (3)$$

$$x_j^{ic} \geq 0 \qquad \forall\, i \in [k],\ c \in [C],\ j \in [n_{ic}] \quad (4)$$

where S_c is the set of small items of class c in S.

Constraint (1) guarantees that the amount of space used in each shelf is at most Δ and constraint (2) guarantees that the amount of space used in each bin is at most B. Constraint (3) guarantees that variables x_j^{ic} are not greater than the total size of small items.

Given a packing $\mathcal{P}$, and a set S of small items, the algorithm SMALL first solves the linear program LPS, and then packs small items in the following way: For each variable x_j^{ic} it packs, while possible, the small items of class c into the shelf N_j^{ic}, so that the total size of the packed small items is at most x_j^{ic}. The possible remaining small items are grouped by classes and packed using the algorithm SFF into new bins. The complexity time of algorithm SMALL is polynomial in n, since the linear program LPS can be solved in polynomial time and the algorithm SFF also has polynomial time.

The following lemma is valid for the algorithm SMALL.

Lemma 4.3.6 *Let $\mathcal{P}$ be a shelf packing of a list of items L, where each bin of $\mathcal{P}$ has at most $\frac{2}{\varepsilon} + 2$ shelves of a same class, G be the set of items in L with size at least ε^2 and S be the set $L \setminus G$. Let G' be a list of items with $G' \preceq G$ and $\hat{\mathcal{P}}$ be a packing of the items $G' \cup S$ obtained from $\mathcal{P}$ as follows:*

1. *Let $\mathcal{P}_1$ be the packing obtained from $\mathcal{P}$ removing the items of S.*
2. *Let $\mathcal{P}_2$ be the packing of G' using the algorithm $\mathrm{A_R}$ over the pair $(\mathcal{P}_1, G')$.*
3. *Let $\hat{\mathcal{P}}$ be the packing obtained applying the algorithm* SMALL *over the pair $(\mathcal{P}_2, S)$.*

Then, we have $|\hat{\mathcal{P}}| \leq (1 + 8C\varepsilon)|\mathcal{P}| + C + 1$.

Proof. Notice that $|\mathcal{P}_2| = |\mathcal{P}|$ and for each shelf N_j in a bin of $\mathcal{P}$, its corresponding shelf N'_j in $\mathcal{P}_2$ is such that $s(N'_j) \leq s(N_j)$. If $|\hat{\mathcal{P}}| = |\mathcal{P}|$ then the lemma follows. So assume that the algorithm SMALL uses additional bins to pack the items of S.

Given a bin E, denote by $ns(E)$ the number of shelves in E, $ns_c(E)$ the number of shelves of class c in E, $ss(E)$ the total size of small items in E and $ss_c(E)$ the total size of small items of class c in E.

Consider the linear program LPS. An optimum solution for LPS leads to an optimal fractional packing $\mathcal{P}^*$ of the small items such that $|\mathcal{P}^*| = |\mathcal{P}|$. Consider a bin P_i^* of $\mathcal{P}^*$ and $\hat{P}_i$ the corresponding bin in $\hat{\mathcal{P}}$. We first prove that the following inequality is valid,

$$ss(P_i^*) - ss(\hat{P}_i) \leq 4C\varepsilon. \tag{4.13}$$

To prove (4.13), notice that $ns_c(P_i^*) = ns_c(\hat{P}_i)$ for each class c. Given a shelf N_j^{ic} and the corresponding variable x_j^{ic}, the algorithm SMALL packs a set of items T_j^{ic} in N_j^{ic} such that $x_j^{ic} - s(T_j^{ic}) \leq \varepsilon^2$ since a small item has size at most ε^2. Since each bin in $\mathcal{P}^*$ has at most $\frac{2}{\varepsilon} + 2$ shelves of a same class, we have for each class $c \in [C]$

$$\begin{aligned} ss_c(\hat{P}_i) &\geq \sum_{j=1}^{n_{ic}} (x_j^{ic} - \varepsilon^2) = ss_c(P_i^*) - ns_c(P_i^*)\varepsilon^2 \\ &\geq ss_c(P_i^*) - (\frac{2}{\varepsilon} + 2)\varepsilon^2 \geq ss_c(P_i^*) - 4\varepsilon. \end{aligned}$$

Since the above inequalities are valid for each class we can conclude the proof of (4.13). From (4.13), we know that the total size of small items packed in additional bins by SMALL with the algorithm SFF, is at most $4C\varepsilon|\mathcal{P}^*| = 4C\varepsilon|\mathcal{P}|$. Denote by $\hat{\mathcal{Q}}$ the set of additional bins. Each shelf generated by the algorithm SFF is filled by at least $\Delta - \varepsilon^2$, except perhaps in C shelves. Therefore, the number of shelves in $\hat{\mathcal{Q}}$ is at most $\left\lceil \frac{4C\varepsilon|\mathcal{P}|}{1-\varepsilon^2} \right\rceil + C \leq 8C\varepsilon|\mathcal{P}| + C + 1$. Since each additional bin has at least one shelf, the number of bins in $\hat{\mathcal{Q}}$ is at most $8C\varepsilon|\mathcal{P}| + C + 1$. Therefore, the number of bins in $\hat{\mathcal{P}}$ is at most $(1 + 8C\varepsilon)|\mathcal{P}| + C + 1$. □

Analysis of the Algorithm $\mathrm{ASBP}''_\varepsilon$

In this section we conclude the analysis of the algorithm $\mathrm{ASBP}''_\varepsilon$. First, let T_{LR} and T_S be the time complexities of algorithms $\mathrm{A_{LR}}$ and SMALL, respectively. Notice that the time complexity of algorithm $\mathrm{ASBP}''_\varepsilon$ is dominated by steps 2–4, that have time complexity $O(T_{\mathrm{LR}} + T_{\mathrm{LR}}T_S)$. Since the time complexity of algorithms $\mathrm{A_{LR}}$ and SMALL is polynomial for fixed ε, the time complexity of algorithm $\mathrm{ASBP}''_\varepsilon$ is also polynomial. The following lemma concludes the analysis of the algorithm $\mathrm{ASBP}''_\varepsilon$.

Lemma 4.3.7 *The algorithm* $\mathrm{ASBP}''_\varepsilon$*, is an APTAS for the* CCSBP *problem when the given instance* I *is such that* $\Delta = 1$, $\varepsilon \leq (d + \Delta)/B$ *and the number of different classes is bounded by some constant* C.

Proof. Given an instance $I = (L, s, c, d, \Delta, B)$, with $\Delta = 1$, let G be the set of items in L with size at least ε^2 and S the set $L \setminus G$.

The items in G are packed by the algorithm $\mathrm{A_{LR}}$. It first partitions G into lists G_c for each class c and then it partitions each list G_c into groups $G_c^1 \succeq G_c^2 \succeq \ldots \succeq G_c^{k_c}$. From Lemma 4.3.5 the following inequality is valid for the list $G^1 = \cup_{c=1}^{C} G_c^1$.

$$|\mathcal{P}_1| \leq 4\varepsilon\,\mathrm{OPT}(I) + 1. \tag{4.14}$$

The packing of the items in $G_1^2\| \ldots \|G_1^{k_1}\| \ldots \|G_C^2\| \ldots \|G_C^{k_C}$ is obtained from the set of all possible packings of $\overline{G_1^1}\| \ldots \|\overline{G_1^{k_1-1}}\| \ldots \|\overline{G_C^1}\| \ldots \|\overline{G_C^{k_C-1}}$. Notice that

$$\overline{G_1^1}\| \ldots \|\overline{G_1^{k_1-1}}\| \ldots \|\overline{G_C^1}\| \ldots \|\overline{G_C^{k_C-1}} \succeq G_1^2\| \ldots \|G_1^{k_1}\| \ldots \|G_C^2\| \ldots \|G_C^{k_C}.$$

Let $\mathcal{O}$ be an optimum shelf packing of I, $\mathcal{O}_1$ the packing obtained from $\mathcal{O}$ without the items of S but with the possible empty shelves and $\mathcal{O}_2$ the packing of $\mathcal{O}_1$ rounding down each item size to the corresponding item in

$$\overline{G_1^1}\| \ldots \|\overline{G_1^{k_1-1}}\|, \ldots, \|\overline{G_C^1}\| \ldots \|\overline{G_C^{k_C-1}}.$$

Clearly, $\mathcal{O}_2 \in \mathbb{P}$, where $\mathbb{P}$ is the set of packings generated by the algorithm $\mathrm{A_{ALL}}$. Let $\hat{\mathcal{O}}$ be a packing obtained from the algorithm $\mathrm{A_R}$ over the pair

$$(\mathcal{O}_2, G_1^2\| \ldots \|G_1^{k_1}\|, \ldots, \|G_C^2\| \ldots \|G_C^{k_C}).$$

If $\mathcal{Q}$ is a packing obtained applying the algorithm SMALL over the pair $(\hat{\mathcal{O}}, S)$, we have from Lemma $4.3.6$ the following result.

$$\mathcal{Q} \leq (1 + 8C\varepsilon)|\mathcal{O}| + C + 1 = (1 + 8C\varepsilon)\mathrm{OPT}(I) + C + 1 \tag{4.15}$$

Since the algorithm $\mathrm{ASBP}''_\varepsilon$ obtains a packing $\mathcal{P}$ that uses at most the number of bins in $\mathcal{P}_1 \cup \mathcal{Q}$, the theorem follows from inequalities (4.14) and (4.15). □

From lemmas 4.3.2 and 4.3.7, the following statement holds.

Theorem 4.3.8 *The algorithm* $\mathrm{ASBP}_\varepsilon$ *is an APTAS for the* CCSBP *problem.*

4.4 Concluding Remarks

In this paper we consider the CCSBP problem, a class constrained bin packing problem with non-null shelf divisions. Although this problem has many practical applications, to our knowledge, this is the first paper to present approximation results for it. We first presented hybrid

versions of the First Fit (Decreasing) and Best Fit (Decreasing) algorithms for the bin packing problem to the CCSBP problem. When the number of different classes of items is bounded by a constant C, we prove that the versions of the First Fit and Best Fit have asymptotic performance bound 3.4 and the versions of the First Fit Decreasing and Best Fit Decreasing have asymptotic performance bound 2.445. We also presented an APTAS for this same case whose running time is

$$O(n^{O(2/\varepsilon)^{O(1/\varepsilon^2)}C/\varepsilon^3}).$$

This algorithm is more of theoretical (rather than practical) interest since it has a high running time (yet polynomial). When the number of classes is not bounded by a constant we show that the algorithm SFFD has absolute performance bound 3.

4.5 Acknowledgements

We would like to thank the anonymous referees for the helpful suggestions which improved the presentation of this paper.

4.6 Bibliography

1. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 2, pages 46–93. PWS, 1997.

2. M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. *First Brazilian Symposium on Graph, Algorithms and Combinatorics. Eletronic Notes in Discrete Mathematics*, 7:1–4, 2001.

3. W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

4. J. S. Ferreira, M. A. Neves, and P. Fonseca e Castro. A two-phase roll cutting problem. *European Journal of Operational Research*, 44(2):185–196, 1990.

5. R. Hoto, M. Arenales, and N. Maculan. The one dimensional compartmentalized cutting stock problem: a case study. *To appear in European Journal of Operational Research*.

6. F. P. Marques and M. Arenales. The constrained compartmentalized knapsack problem. *To appear in Computer & Operations Research*.

7. H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, 4(6):313–338, 2001.

8. H. Shachnai and T. Tamir. Tight bounds for online class-constrained packing. *Theoretical Computer Science*, 321(1):103–123, 2004.

9. D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Res. Logistics*, 41:579–585, 1994.

10. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

11. E. C. Xavier and F. K. Miyazawa. Approximation schemes for knapsack problems with shelf divisions. *Theoretical Computer Science*, 352(1–3):71–84, 2006.

Capítulo 5

Artigo: *A Note on Dual Approximation Algorithms for Class Constrained Bin Packing Problems*

E. C. Xavier[1] F. K. Miyazawa[2]

Abstract

In this paper we present a dual approximation scheme for the class constrained shelf bin packing problem. In this problem, we are given bins of capacity 1, and n items of Q different classes, each item e with class c_e and size s_e. The problem is to pack the items into bins, such that items of different classes must be packed in different shelves, inside the bin, that are separated by non-null shelf divisions. We also present a dual approximation scheme for the class constrained bin packing problem. In this problem, items must be packed in such a way that each bin contains at most C different classes and has total items size at most 1. A dual approximation scheme may produce infeasible packings but only within a small tolerance.

Key Words: Bin Packing, Approximation Algorithms.

5.1 Introduction

In this paper we study class constrained bin packing problems, that are generalizations of the well known NP-hard bin packing problem. We first consider the class constrained shelf bin

[1]Corresponding author: para@ic.unicamp.br — Instituto de Computação — Universidade Estadual de Campinas, Caixa Postal 6176 — 13084–971 — Campinas–SP — Brazil — Phone (+55)(19) 3788-5882.

[2]fkm@ic.unicamp.br — Instituto de Computação — Universidade Estadual de Campinas, Caixa Postal 6176 — 13084–971 — Campinas–SP — Brazil — Phone (+55)(19) 3788-5882.

packing (CCSBP) problem. In this problem we are given a tuple $I = (L, s, c, Q, d, \Delta)$, where $L = (a_1, \ldots, a_n)$ is a list of n items, each item $a_i \in L$ with size $0 < s_{a_i} \leq 1$ and class $c_{a_i} \in \{1, \ldots, Q\}$, d is the size of a shelf division and Δ is the maximum size of a shelf. We are also given a set of bins, each one with capacity 1.

Given a list or set of items S we denote by $s(S)$ the total size of items in S, *i.e.* $s(S) = \sum_{e \in S} s_e$.

A *shelf packing* $\mathcal{P}$ of an instance I for the CCSBP problem is a packing of the items in a set of bins $\mathcal{P} = \{P_1, \ldots, P_k\}$, where the items packed in a bin $P_i \in \mathcal{P}$ are partitioned into shelves $\{S_1^i, \ldots, S_{q_i}^i\}$ such that for each shelf S_j^i we have that $s(S_j^i) \leq \Delta$, all items in S_j^i are of the same class and $\sum_{j=1}^{q_i}(s(S_j^i) + d) \leq 1$. The problem is to find a shelf packing that uses the minimum number of bins.

We also consider the class constrained bin packing problem, which we denote by CCBP. In this problem we are given a tuple $I = (L, s, c, C, Q)$ where $L = (a_1, \ldots, a_n)$ is a list of n items, each item $a_i \in L$ with size $0 < s_{a_i} \leq 1$ and class $c_{a_i} \in \{1, \ldots, Q\}$, and a set of bins, each one with capacity 1 and C compartments. A packing for instance I is a set of bins $\mathcal{P} = \{P_1, \ldots, P_k\}$ such that the number of different classes of items packed in each bin P_i is at most C and the total items size in each bin is at most 1. The problem is to find a packing of instance I that uses the minimum number of bins.

In both problems we assume that Q, the number of different classes in the input instance, is bounded by a constant.

Given an algorithm $\mathcal{A}$ for the CCBP or CCSBP problem and an instance I, we denote by $\mathcal{A}(I)$ the number of bins used by the algorithm to pack this instance. We denote by $\mathrm{OPT}(I)$ the number of bins used by an optimum solution to pack the instance I. In both notations the problem considered will be clear from the context. Given an integer t, we denote by $[t]$ the set $\{1, \ldots, t\}$.

In [5], Hochbaum and Shmoys presented the concept of dual approximation algorithms where one has to find an infeasible optimal solution, and the quality of the algorithm is measured by how infeasible is the generated solution. There are some cases where the restrictions of the problem are flexible in practice and the concept of dual approximation algorithms can be applied.

A dual polynomial time approximation scheme (dual PTAS) for the CCSBP problem is an algorithm that, for all instances I, produces solutions that use at most $\mathrm{OPT}(I)$ bins, each bin with size at most $(1 + O(\epsilon))$ and each shelf of the bin with size at most $(1 + O(\varepsilon))\Delta$. A dual PTAS for the CCBP problem is an algorithm that, for all instances I, produces solutions that use at most $\mathrm{OPT}(I)$ bins, each bin with size at most $(1 + O(\epsilon))$. In both cases ε is a fixed parameter given to the algorithm.

Packing problems with class constraints have many applications in multimedia storage systems, resource allocation [15, 11, 4, 7, 14, 16, 13, 3, 19] and in manufacturing systems [6, 9, 1].

The CCSBP problem appears in the iron and steel industry [2, 8, 10, 17, 18].

The CCSBP problem admits an asymptotic polynomial time approximation scheme [17]. A knapsack version of this problem also admits a PTAS [18]. This paper is the first one to present a dual PTAS for the CCSBP problem.

We also present a dual PTAS for the CCBP problem. Notice that a dual approximation scheme for the CCBP problem was first presented by Shachnai and Tamir [12] also considering that the number of different classes in the input instance is bounded by a constant. The complexity time of their algorithm is $O(n^{16Q/\varepsilon^2})$. In their paper they presented a dual PTAS using techniques that group small items together. They also said: "We cannot adopt the technique commonly used for packing, where we first consider large items and then add the small items". In this paper we show how to adopt the traditional technique and obtain a dual PTAS with an easier analysis, also considering that Q is a fixed constant. Although the easier analysis, the complexity time of our algorithm is $O(Tn^{O(2^Q Q(\log_{1+\varepsilon} 1/\varepsilon)^{1/\varepsilon})})$, where T is the complexity time to solve a linear program (see Section 5.3).

In section 5.2 we present a dual PTAS for the CCSBP problem using traditional techniques, and linear programming to pack small items. In section 5.3 we use these ideas to obtain a dual PTAS for the CCBP problem. The analysis of it is easier than the one presented by Shachnai and Tamir [12].

5.2 A dual PTAS for the CCSBP Problem

In this section we present a dual PTAS for the CCSBP problem.

Let $I = (L, s, c, Q, d, \Delta)$ be an instance for the CCSBP problem. We first present a dual PTAS for the case where the maximum size of a shelf plus the shelf divisor satisfy $\Delta + d \leq \varepsilon$.

Hochbaum and Shmoys [5] presented a dual PTAS, which we denote by $\mathcal{A}_{HS}$, for the classical bin packing problem. Consider an algorithm that constructs a list of shelves S in a straightforward manner: For each class, it packs the items of this class using the algorithm $\mathcal{A}_{HS}$ considering shelves as bins, each one with size Δ. Since the algorithm $\mathcal{A}_{HS}$ is a dual PTAS the number of generated shelves by the algorithm is at most the number of shelves used in any optimal solution, which we denote by $\mathrm{OPT}(I)_s$. Moreover each generated shelf has size at most $(1 + \varepsilon)\Delta$.

Given the list of shelves S, consider another algorithm that packs the shelves in the following manner: It packs shelves (including the shelf divisors) in a bin until for the first time the total size of packed shelves becomes greater than 1. Then it proceeds with a new bin. It is easy to prove the following result for this algorithm.

Theorem 5.2.1 *The presented algorithm is a dual PTAS for the CCSBP problem restricted to instances where $\Delta + d \leq \varepsilon$.*

Proof. Notice that the algorithm packs all items in at most $\mathrm{OPT}(I)_s$ shelves and each shelf has its size increased by a factor of at most ε. The total size of items and shelf divisors that the algorithm has to pack into bins is

$$s(L) + d\mathrm{OPT}(I)_s \leq \mathrm{OPT}(I).$$

Since the algorithm generated bins with size greater than 1, the algorithm packs all shelves in at most $\mathrm{OPT}(I)$ bins. Since each shelf has size at most 2ε, each generated bin has size at most $(1 + 2\varepsilon)$. □

On the remaining of this section we assume that $\Delta + d \geq \varepsilon$. Notice that the maximum number of shelves completely filled, that can be packed in a bin is at most $\left\lceil \frac{1}{d+\Delta} \right\rceil$, that is at most $\frac{1}{\varepsilon} + 1$. Observe that if there is any bin with more than $\frac{2}{\varepsilon} + 2$ shelves of a same class, it has at least two shelves of this class with total size at most Δ. In this case, these two shelves can be combined into only one shelf. Without loss of generality we assume that each bin in a solution for the CCSBP problem, contains at most $\frac{2}{\varepsilon} + 2$ shelves of a same class.

Throughout the remaining of this section, we assume that s_e for each $e \in L$, d, Δ and the size of the bins are rescaled, such that $\Delta = 1$. We denote by B the new size of the bins.

Let L_b be the list of items with size greater than or equal to ε^2 (big items) and let L_s be the remaining items in L (small items). We round down each item in L_b as follows: each item $e \in L_b$ with size in the interval $[\varepsilon^2(1+\varepsilon^2)^i, \varepsilon^2(1+\varepsilon^2)^{i+1})$ has its size rounded down to $\varepsilon^2(1+\varepsilon^2)^i$, for $i \geq 0$. The rounded items have at most $M = \lceil \log_{(1+\varepsilon^2)} 1/\varepsilon^2 \rceil$ different sizes.

Lemma 5.2.2 *Let $I = (L, s, c, Q, d, \Delta)$ be an instance of the CCSBP problem where $L = L_b \cup L_s$, $\Delta = 1$, the number of distinct items sizes in L_b is at most a constant M, the number of different classes is bounded by a constant Q, each item $e \in L_b$ has size $s_e \geq \varepsilon^2$ and $L_s = L \setminus L_b$. Then there exists a polynomial time algorithm that generates all possible shelf packings of L, removing small items of the packing, with at most $\frac{2}{\varepsilon} + 2$ shelves of a same class in each bin. Moreover, each bin of each generated packing has an indication of the possible shelves that may be used by further small items.*

Proof. The maximum number of big items that can be packed in a shelf is bounded by $p = 1/\varepsilon^2$. Given a class, the number of different shelves for it is bounded by $r' = \binom{p+M+1}{p}$, including a empty shelf that can be used latter to pack only small items. The number of different shelves can be bounded by $r = Qr'$. Since the number of shelves in a bin is bounded by $q = Q(\frac{2}{\varepsilon} + 2)$, the number of different bins is bounded by $u = \binom{q+r}{q}$. Notice that u is a (large) constant since all the values p, q, r and u depends only on ε, Q and M which are constants.

Therefore, the number of all feasible packings is bounded by $\binom{n+u}{n}$, which is bounded by $(n+u)^u$, which in turn is polynomial in n. □

In each generated packing, we then consider the original sizes of the big items, and in this case, the total size of each shelf increases by a factor of at most ε^2. Since the maximum

number of shelves in a bin is bounded by $Q(\frac{2}{\varepsilon}+2)$, the size of each bin increases to at most $B+Q(\frac{2}{\varepsilon}+2)\varepsilon^2$.

The algorithm generates a set, which we denote by $\mathbb{P}$, of all possible packings of the rounded big items. For each of these packings we then consider the big items with their original size. For each packing in $\mathbb{P}$, the algorithm then packs small items using a solution from a linear program. Let $\mathcal{P}=\{P_1,\ldots,P_k\}$ be a shelf packing of a list of items L_b and suppose we have to pack a list L_s of small items, with size at most ε^2, into $\mathcal{P}$. The packing of the small items is obtained from a solution of a linear program. Let $N_i \subseteq \{1,\ldots,Q\}$ be the set of possible classes that are packed in the bin P_i and let $S_1^{ic},\ldots,S_{n_{ic}}^{ic}$ be the shelves of class $c \in N_i$ in the bin P_i of the packing $\mathcal{P}$. For each shelf S_j^{ic}, define a non-negative variable x_j^{ic}. The variable x_j^{ic} indicates the total size of small items of class c that is to be packed in the shelf S_j^{ic}. Denote by $s(S_j^{ic})$ the total size of big items already packed in the shelf S_j^{ic}. Consider the following linear program denoted by LPS1:

$$
\begin{aligned}
& \max \sum_{i=1}^{k}\sum_{c\in N_i}\sum_{j=1}^{n_{ic}} x_j^{ic} & & & \\
s(S_j^{ic}) + x_j^{ic} &\leq (1+\varepsilon^2)\Delta & \forall\, i\in[k],\ c\in N_i,\ j\in[n_{ic}], & \quad (1) & \\
\sum_{c\in N_i}\sum_{j=1}^{n_{ic}} (s(S_j^{ic}) + x_j^{ic} + d) &\leq (1+x\varepsilon^2)B & \forall\, i\in[k], & \quad (2) & \text{(LPS1)}\\
\sum_{i=1}^{k}\sum_{j=1}^{n_{ic}} x_j^{ic} &\leq s(L_s^c) & \forall\, c\in[Q], & \quad (3) & \\
x_j^{ic} &\geq 0 & \forall\, i\in[k],\ c\in[N_i],\ j\in[n_{ic}] & \quad (4) &
\end{aligned}
$$

where L_s^c is the set of small items of class c in L_s.

Constraint (1) guarantees that the amount of space used in each shelf is at most $(1+\varepsilon^2)\Delta$ and constraint (2) guarantees that the amount of space used in each bin is at most $(1+x\varepsilon^2)B$, where $x=Q(\frac{2}{\varepsilon}+2)$. Constraint (3) guarantees that variables x_j^{ic} are not greater than the total size of small items. The number of variables in LPS1 is bounded by $O(nQ2/\varepsilon)$ and the number of constraints is bounded by $O(nQ2/\varepsilon+n+Q)$.

Given a packing $\mathcal{P}$, and a list L_s of small items, the algorithm first solves the linear program LPS1, and then packs small items in the following way: For each variable x_j^{ic} the algorithm packs, while possible, small items of class c into shelf S_j^{ic} of the bin P_i, so that the total size of the packed small items is at most $x_j^{ic}+\varepsilon^2$.

The algorithm returns a packing that uses the minimum number of bins and that packs all items in bins of size at most $(1+(\frac{2}{\varepsilon}+2)2\varepsilon^2 Q)B$.

Since Q and ε are constants, the size of $\mathbb{P}$ is bounded by a polynomial in n. Since the complexity time to solve LPS1 is polynomial, the presented algorithm has a polynomial time complexity. Now we conclude with the following theorem.

Theorem 5.2.3 *The presented algorithm is a dual PTAS for the* CCSBP *problem when* $\Delta+d \geq \varepsilon$.

Proof. Let $O = \{P_1^*, \ldots, P_k^*\}$ be an optimal packing for an instance I of the CCSBP problem (notice that $\mathrm{OPT}(I) = k$). Round down the big items according to the rounding we have presented and remove the small items of O obtaining another packing O'. Clearly $O' \in \mathbb{P}$ and has an indication of the shelves of small items that were packed on it. When the algorithm packs the big items with their original size, the size in each shelf of O' increases by at most ε^2. Since in the linear programming formulation we consider the size of each shelf as $(1+\varepsilon^2)$, there is enough room to pack all small items. So the variables x sums to the total size of small items. We also consider the size of each bin in the linear programming formulation as $(1+(\frac{2}{\varepsilon}+2)\varepsilon^2 Q)B$, so there is enough room to pack all shelves.

During the packing of the small items we increase the size of each shelf by at most ε^2. Since the maximum number of shelves in a bin is $(\frac{2}{\varepsilon}+2)Q$ then the total size of each bin is increased to at most $B + (\frac{2}{\varepsilon}+2)2\varepsilon^2 Q \leq (1+(\frac{2}{\varepsilon}+2)2\varepsilon^2 Q)B$. □

5.3 A dual PTAS for the CCBP Problem

In this section we present a dual PTAS for the CCBP problem using the same ideas of the previous section. This dual PTAS has an easier analysis than the one presented by Shachnai and Tamir [12].

Let L_b be the set of items in L with size at least ε (big items) and let L_s be the remaining items in L (small items). We round down each item in L_b as follows: each item $e \in L_b$ with size in the interval $[\varepsilon(1+\varepsilon)^i, \varepsilon(1+\varepsilon)^{i+1})$ has its size rounded down to $\varepsilon(1+\varepsilon)^i$, for $i \geq 0$. The rounded items have at most $M = \lceil \log_{(1+\varepsilon)} 1/\varepsilon \rceil$ different sizes.

It is not hard proof the following lemma that is similar to Lemma 5.2.2.

Lemma 5.3.1 *Let* $I = (L, s, c, C, Q)$ *be an instance of the* CCBP *problem where* $L = L_b \cup L_s$*, the number of distinct items sizes in* L_b *is at most a constant* M*, the number of different classes is bounded by a constant* Q*, each item* $e \in L_b$ *has size* $s_e \geq \varepsilon$*, and* $L_s = L \setminus L_b$*. Then there exists a polynomial time algorithm that generates all possible packings of* L *removing the small items of the packing. Moreover, each bin of each generated packing has an indication of the possible classes that may be used to pack the small items.*

Proof. The number of big items that can be packed in a bin is bounded by $y = 1/\varepsilon$. The number of distinct types of big items is bounded by MQ. The number of different configurations of bins is bounded by $r' = \binom{y+MQ+1}{y}$, including the empty bin. If we also consider additional classes to pack small items in each configuration, the number of different configurations is bounded by

$r = r'2^Q$, which is a constant. Notice that we only consider configurations that satisfy the class constraints.

The number of all feasible packings is bounded by $\binom{n+r}{n}$, which is bounded by $(n+r)^r$, which in turn is polynomial in n. □

We then consider the original size of the items in each of the generated packings. In this case, the size of each bin increases by at most a factor of ε.

The algorithm generates a set, which we denote by $\mathbb{P}$, of all possible packings of the big items. For each one of these packings the algorithm packs the small items in the following way: Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a packing of the list of items L_b and suppose we have to pack a list L_s of small items, with size at most ε, into $\mathcal{P}$. The packing of the small items is obtained from a solution of a linear program. Let $N_i \subseteq \{1, \ldots, Q\}$ be the set of possible classes that may be used to pack the small items in the bin P_i of the packing $\mathcal{P}$. For each class $c \in N_i$, define a non-negative variable x_c^i. The variable x_c^i indicates the total size of small items of class c to be packed in the bin P_i. Denote by $s(P_i)$ the total size of big items already packed in the bin P_i. Consider the following linear program denoted by LPS2:

$$\max \sum_{i=1}^{k} \sum_{c \in N_i} x_c^i$$

$$\begin{aligned}
s(P_i) + \sum_{c \in N_i} x_c^i &\leq (1+\varepsilon) && \forall\, i \in [k] && (1)\\
\sum_{i=1}^{k} x_c^i &\leq s(L_s^c) && \forall\, c \in [Q], && (2)\\
x_c^i &\geq 0 && \forall\, i \in [k], c \in [N_i], && (3)
\end{aligned} \qquad \text{(LPS2)}$$

where L_s^c is the set of small items of class c in L_s.

Constraint (1) guarantees that the items packed in each bin satisfy its capacities and constraint (2) guarantees that the total use of variables x_c^i is not greater than the total size of small items for each class c. In this linear program, the number of variables is bounded by nQ and the number of constraints is bounded by $n + Q$.

Given a packing $\mathcal{P}$, and a list L_s of small items, the algorithm first solves the linear program LPS2, and then packs small items in the following way: For each variable x_c^i, it packs, while possible, the small items of class c into the bin P_i, so that the total size of the packed small items is at most $x_c^i + \varepsilon$.

The algorithm returns a packing that uses the minimum number of bins and that packs all items in bins of size at most $(1 + (C+1)\varepsilon)$. The number of packings in the set $\mathbb{P}$ can be bounded by $T_1 = O(n^{2^Q Q(\log_{1+\varepsilon} 1/\varepsilon)^{1/\varepsilon}})$. Let T_2 be the worst complexity time to solve a linear program LPS2. The complexity time of the entire algorithm can be bounded by $O(T_1 T_2)$, which is polynomial since Q and ε are constants and the complexity time T_2 is polynomial.

We conclude with the following theorem.

Theorem 5.3.2 *The presented algorithm is a dual PTAS for the* CCBP *problem.*

Proof. Let $O = \{P_1^*, \ldots, P_k^*\}$ be an optimal packing for an instance I of the CCBP problem. Round down the big items according to the rounding we have presented and remove the small items of O obtaining another packing O'. Clearly $O' \in \mathbb{P}$ and has an indication of the classes of small items that were packed on it. When the algorithm packs the big items with their original size, the size of each bin in O' increases by at most ε. Since in the linear programming formulation we consider the size of each bin as $(1 + \varepsilon)$, there is enough room to pack all small items. So the variables x sums to the total size of small items.

During the packing of the small items we increase the size of each bin by at most ε for each class in the bin. So the total size of each bin is increased to at most $(1 + (C + 1)\varepsilon)$. □

5.4 Bibliography

1. M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. In *Proceedings of the 1th Brazilian Symposium on Graph Algorithms and Combinatorics*, volume 7 of *Electronic Notes in Discrete Mathematics*, 2001.

2. J. S. Ferreira, M. A. Neves, and P. Fonseca e Castro. A two-phase roll cutting problem. *European J. Operational Research*, 44:185–196, 1990.

3. S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continous media servers. *Parallel Computing Journal*, 24(1):91–122, 1998.

4. L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proceedings of SODA*, pages 223–232, 2000.

5. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for schedulling problems: practical and theoretical results. *journal of the ACM*, 34(1):144–162, 1987.

6. J. R. Kalagnanam, M. W. Dawande, M. Trumbo, and H. S. Lee. The surplus inventory matching problem in the process industry. *Operations Research*, 48(4):505–516, 2000.

7. S. R. Kashyap and S. Khuller. Algorithms for non-uniform size data placement on parallel disks. In *Proceedings of FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 265–276, 2003.

8. F. P. Marques and M. Arenales. The constrained compartmentalized knapsack problem. *To appear in Computer & Operations Research*.

9. M. Peeters and Z. Degraeve. The co-printing problem: A packing problem with a color constraint. *Operations Research*, 52(4):623–638, 2004.

10. M. Arenales R. Hoto and N. Maculan. The one dimensional compartmentalized cutting stock problem: a case study. *To appear in European Journal of Operational Research*.

11. H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29:442–467, 2001.

12. H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, 4(6):313–338, 2001.

13. H. Shachnai and T. Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, 32(4):651–678, 2002.

14. H. Shachnai and T. Tamir. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In *Proceedings of 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, RANDOM-APPROX*, volume 2764 of *Lecture Notes in Computer Science*, pages 165–177, 2003.

15. H. Shachnai and T. Tamir. Tight bounds for online class-constrained packing. *Theoretical Computer Science*, 321(1):103–123, 2004.

16. J. L. Wolf, P. S. Wu, and H. Shachnai. Disk load balancing for video-on-demand-systems. *ACM Multimedia Systems Journal*, 5:358–370, 1997.

17. E. C. Xavier and F. K. Miyazawa. A one-dimensional bin packing problem with shelf divisions. In *2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics*, volume 19 of *Electronic Notes in Discrete Mathematics*, 2005.

18. E. C. Xavier and F. K. Miyazawa. Approximation schemes for knapsack problems with shelf divisions. *Theoretical Computer Sciense*, 352(1-3):71–84, 2006.

19. E. C. Xavier and F. K. Miyazawa. The class constrained bin packing problem with applications to video-on-demand. In *Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON'06)*, volume 4112 of *Lecture Notes in Computer Science*, pages 439–448, 2006.

Capítulo 6

Artigo: *The Class Constrained Bin Packing Problem with Applications to Video-on-Demand*

E. C. Xavier[2] F. K. Miyazawa[2]

Abstract

In this paper we present approximation results for a class constrained bin packing problem that has applications to Video-on-Demand Systems. In this problem we are given bins of capacity B with C compartments, and n items of Q different classes, each item $i \in \{1, \ldots, n\}$ with class c_i and size s_i. The problem is to pack items into bins, where each bin contains at most C different classes and has total items size at most B. We present several approximation algorithms for offline and online versions of the problem. The presented results are the best known to the author's knowledge.

Key words: Bin Packing, Video-on-Demand.

6.1 Introduction

In this paper we study the class constrained version of the well known bin packing problem, which we denote by CCBP (Class Constrained Bin Packing). In this problem we are given a tuple $I = (L, s, c, C, Q)$ where $L = (a_1, \ldots, a_n)$ is a list of items, each item $a_i \in L$ with

[1]A preliminary version of this paper appeared as an extended abstract in COCOON 2006, LNCS 4112, pp. 439–448, 2006.

[2]Instituto de Computação — Universidade Estadual de Campinas, Caixa Postal 6176 — 13084–971 — Campinas–SP — Brazil, {eduardo.xavier,fkm}@ic.unicamp.br.

size $0 < s_{a_i} \leq B$ and class $c_{a_i} \in \{1, \ldots, Q\}$, and a set of bins, each one with capacity B and C compartments. A packing $\mathcal{P}$ of L is a partition of the items into bins, where each bin has total items size at most B and the number of different classes in each part is at most C. The problem is to find a packing of L into the minimum number of bins. In the online version of the CCBP problem the items must be packed in the order $(a_1, \ldots, a_n)$, where each item a_i must be packed without knowledge of further items. We consider that $1 < C < Q$, otherwise the CCBP problem can be solved as the original bin packing, since if $C = 1$ then items of different classes must be packed in different bins and if $C \geq Q$ then the class constraints are irrelevant. We also consider the version of this problem with bins of different sizes. In this case we have T different bin sizes. The input instance is a tuple $I = (L, s, c, w, C, Q)$ where $w : \{1, \ldots, T\} \rightarrow \mathbb{R}^+$ is a function of bins size. We assume w.l.o.g that for each $i \in \{1, \ldots, T\}$, $w(i) \leq B$. In this case, the problem is to pack all items into bins such that the total size of used bins is minimized. This problem is denoted by VCCBP (Variable Class Constrained Bin Packing). Packing problems with class constraints have many applications in multimedia storage systems, resource allocation [23, 19, 8, 13, 22, 9, 21, 7] and in operations research like manufacturing systems [12, 17, 5, 26, 27].

6.1.1 Notation

In the online case, the bins used to pack the items are classified as *open* or *closed*. An empty bin is declared open when it receives its first item, and remains so until it is declared closed. Only open bins may receive items. Once a bin is closed, it cannot be declared open again. We consider the bounded and unbounded space versions for the online CCBP problem. In the l-bounded space problem an algorithm must keep at any time during its execution at most l open bins. In the unbounded version an algorithm may keep an unbounded number of open bins.

Given an algorithm $\mathcal{A}$ for the CCBP problem and an instance I, we denote by $\mathcal{A}(I)$ the number of bins used by the algorithm to pack this instance. We denote by $\mathrm{OPT}(I)$ the number of bins used by an optimum (offline) solution to pack the instance I. The algorithm $\mathcal{A}$ has an absolute approximation factor α, if for every I it satisfies $\mathcal{A}(I) \leq \alpha\mathrm{OPT}(I)$. It has an α approximation factor if for every I, the algorithm produces a solution such that $\mathcal{A}(I) \leq \alpha\mathrm{OPT}(I) + \beta$ where β is a constant. Given an algorithm A_ε, for some $\varepsilon > 0$, and an instance I for some problem P we denote by $A_\varepsilon(I)$ the value of the solution returned by algorithm A_ε when executed on instance I. We say that A_ε, for $\varepsilon > 0$, is an asymptotic polynomial time approximation scheme (APTAS) for the problem CCBP if there exist constants t and β such that $A_\varepsilon(I) \leq (1 + t\varepsilon)\mathrm{OPT}(I) + \beta$ for any instance I. An online algorithm $\mathcal{A}$ for a minimization problem is said to have a competitive ratio α if there exists a constant β such that $\mathcal{A}(I) \leq \alpha\mathrm{OPT}(I) + \beta$ for any instance I.

Let I be an instance of the CCBP problem and L be the list of items in I. We write that

$a \in I$ with the same meaning of $a \in L$, and we denote $s(I) = s(L) = \sum_{a \in L} s(a)$. Given an integer M, we denote by $[M]$ the set $\{1, \ldots, M\}$.

Given two sequences $L_a = (a_1, \ldots, a_n)$ and $L_b = (b_1, \ldots, b_m)$, we denote the concatenation of these two lists by $L_a \| L_b$, i.e, $L_a \| L_b = (a_1, \ldots, a_n, b_1, \ldots, b_m)$. Given a packing $\mathcal{P}$ we denote by $|\mathcal{P}|$ the number of bins in $\mathcal{P}$.

Throughout this paper, we use the terms color and class with the same meaning. We say that a bin is *colored* if it contains items of C different classes. In this case, this bin cannot pack any other item of a different class. A bin is said to be *full* if the total size of the items packed inside it is equal to B.

6.1.2 Related Work

A special case of the CCBP problem is the Bin Packing problem, which is one of the most studied problems in the literature. Some of the most famous algorithms for the bin packing problem are the algorithms FF, BF, FFD and BFD, with asymptotic performance bounds $17/10$, $17/10$, $11/9$ and $11/9$, respectively. We refer the reader to Coffman *et al.* [2] for a survey on approximation algorithms for bin packing problems. Fernandez de la Vega and Lueker [6] presented an APTAS for the bin packing problem. The online bin packing is also a well studied problem. There are many online algorithms presented in the literature for the bin-packing problem. The algorithms FF, NF, and BF are online and were investigated by Ullman [24], Johnson [10] and Johnson *et al.* [11]. Subsequent papers proposed algorithms with better approximation ratios that pack items according to interval sizes. Yao [28], and Lee and Lee [15] presented the Harmonic and Refined Harmonic algorithms with competitive ratio 1.692 and 1.636 respectively. To our knowledge the best online algorithm, with a competitive ratio of 1.58889, was presented by Seiden [18]. The best lower bound for this problem is 1.54014 due to van Vliet [25]. Recently the class-constrained versions of packing problems have obtained attention. In [5, 4], Dawande *et al.* claimed to present an approximation scheme for the offline VCCBP problem when the number of different classes Q in the input instance is bounded by a constant. In [20], Shachnai and Tamir presented a dual polynomial time approximation scheme for the offline class constrained bin packing problem (CCBP). They also consider that the number of different classes in the input instance is bounded by a constant. In this case, given an instance I, the problem is to find a packing of the items in at most $\mathrm{OPT}(I)$ bins, each bin with size at most $(1 + O(\varepsilon))B$. In [19], Shachnai and Tamir presented theoretical results for a Multiple Knapsack problem with class constraints where all items have unit size. They introduce this problem with applications to video-on-demand servers. Subsequently to this work, Golubchik *et al.* [8] presented an approximation scheme to the problem. Later, Kashyap and Khuller [13], also presented approximation schemes to the problem, but they consider that the class requirement of items are not equal to all classes. Shachnai and Tamir in [23], presented algorithms

for the online CCBP problem when all items have equal size. In this case they provide a lower bound of 2 to the problem and also algorithms that get a competitive ratio of 2.

6.1.3 Results

In this paper we generalize the work presented by Shachnai and Tamir [23], since we consider the online CCBP problem where items can have different sizes. We show that the bounded space online CCBP problem cannot have a constant competitive ratio. Moreover if any item of the instance have size at least $\varepsilon < B$ we show that there is no algorithm with competitive ratio better than $\mathrm{O}(1/C\varepsilon)$. For the unbounded space problem we present an online algorithm with competitive ratio in $[2.666, 2.75]$. We also present some results for the offline problem. When all items have equal size, we present an $(1 + 1/C)$-approximation algorithm. When items have size at most B/m, for some integer m, we show an algorithm with approximation factor $(1 + 1/C + 1/\min\{C, m\})$. Notice that we consider that the number of different classes Q is part of the input in these cases. We implemented these practical algorithms and we also present in this paper some experimental results for them. The experiments show that the algorithms generate solutions of high quality and can be used in practice. The VCCBP problem was first considered by Dawande *et al.* [5, 4] where a tentative of an APTAS was considered when Q is bounded by a constant. We observed that their algorithm does not lead to an APTAS as claimed. First of all, they do a linear rounding step of the list of items L and then obtain an optimal packing for the new list. Doing this they do not guarantee a packing for the original items because of the class constraints. To pack the small items they use a First Fit strategy, and claim that each bin (at most a constant number of bins), is filled by at least $(1-O(\varepsilon))$, but this is also not true due to the class constraints. In this paper we show the points where their algorithm fails and present an APTAS for the VCCBP problem for fixed Q. In the linear rounding step we separate items by colors and generate all possible packings for the rounded items. To pack the small items we use another strategy.

Organization: In Section 6.2 we present the application of the CCBP problem to data placement of videos. In Section 6.3, motivated by the video-on-demand systems applications, we present practical approximation algorithms for the CCBP problem considering that all items have equal size. In Section 6.4, we present lower bounds for the competitive ratio of any algorithm for the bounded space online CCBP problem. In this section, we also present online algorithms, one of them with competitive ratio in $[2.666, 2.75]$. In Section 6.5 we present an APTAS for the VCCBP problem when Q is bounded by a constant. In Section 6.6 we show experimental results of the practical algorithms shown in Section 6.3.

6.2 Applications of the CCBP Problem to Data Placement on Video-on-Demand Servers

The first work to consider packing problems with class constraints as a data placement problem was the one of Shachnai and Tamir [19]. They considered the knapsack version of the CCBP problem. In this case N bins are given, and the objective is to pack the maximum number of items satisfying the class constraints in each bin. Suppose we have a server of videos with N disks, each disk $j \in \{1, \ldots, N\}$ with *storage capacity* C_j and *load capacity* B_j. That is, each disk j can store C_j movies and can attend at most B_j simultaneously requests for videos. The problem is to construct a server such that, based on expected requests for movies (computed by movies popularity), the number of attended requests is maximized. The total load capacity of the server is $B_T = \sum_{j=1}^{N} B_j$. The movies considered to be stored in the server are $F_1, F_2, \ldots, F_f$ with popularity parameters $p_1, p_2, \ldots, p_f$, where $\sum_{i=1}^{f} p_i = 1$. Given these popularity parameters we compute expected requests for each movie at any time. These expected requests are, for each i, defined as $r_i = B_T p_i$. Notice that $\sum_{i=1}^{f} r_i = B_T$ (we suppose that each r_i is an integer).

Consider for example that we have a server with two hard disks. Disk 1 has $C_1 = 2$ and $B_1 = 4$ and disk 2 has $C_2 = 2$ and $B_2 = 8$. There are three movies F_1, F_2 and F_3, with popularity parameters $p_1 = 1/4$, $p_2 = 1/4$ and $p_3 = 1/2$. Computing the expected requests one obtain $r_1 = 3$, $r_2 = 3$ and $r_3 = 6$. One optimal solution is given in Figure 6.1. One copy of movie F_1 is done in disk 1, a copy of movie F_2 is done in disk 1 and 2, and a copy of movie F_3 is done in disk 2. Notice that not all load capacity of the disks can be used. We call a perfect placement when all load capacity is used, i.e, all requests are allocated.

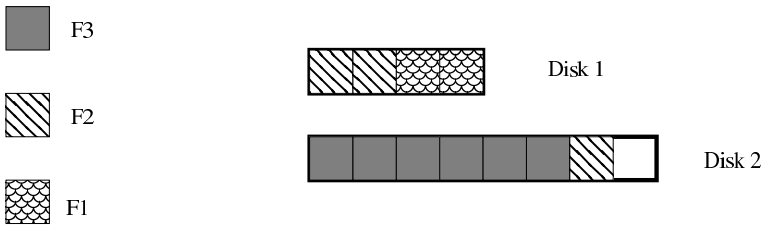

Figure 6.1: An optimal solution for the given video server.

This problem was shown to be NP-hard by Shachnai and Tamir [19]. Golubchik *et al.* [8] show that even if all disks are equal, i.e, have the same load and store capacities, the problem remains NP-hard.

We can also consider the following problem: given a set of requests for a set of movies, construct a server using the minimum number of disks. This problem is NP-hard since, given an instance for the data placement with N disks, a perfect placement exists, if and only if we

can find a packing for all requests using at most N disks. When all disks are equal, we can see this data placement problem as a special case of the CCBP problem. In this case we have an instance $I = (L, s, c, C, Q)$, where each item $i \in L$ is a request for a load of class $c_i \in Q$ (the movie type). All items have the same size and C is the capacity of the disks, i.e, the number of different movies that the disk can store. That is, we want to construct a video server storing the videos and distributing all the requests minimizing the number of used disks.

6.3 Practical Approximation Algorithms

In this section we consider the problem where all items have unit size. As we saw, this problem is NP-hard and has applications in the data placement problem for video-on-demand. In this case, we can consider that items are given as a list of sets $U_1, \ldots, U_Q$, where each set U_i has n_i items of unit size with class i. Each bin packs at most B items of at most C different sets. The problem is to pack all sets of items in the minimum number of bins. We say that a set of items is *totally packed* in a bin if all of its items are packed in the bin, otherwise we say that a set is *partially packed*. We also say that a bin packs entirely C sets, if C sets are totally packed in the bin.

We adapt here, an algorithm known as Moving-Window (MW) first presented by Shachnai and Tamir [19] and also used later by Golubchik *et al.* [8] and Kashyap and Khuller [13]. In these previous works the algorithm was considered for the knapsack version of the problem, where one must have to pack the maximum number of items in a given number of bins.

Moving-Window (MW**):** The algorithm keeps a vector $R = (R[1], R[2], \ldots, R[Q])$ representing non-packed items in such a way that $R[i]$ is the number of remaining items to be packed of some set U_j. The vector is maintained in non-decreasing order of the values $R[i]$ during all the execution of the algorithm. If at any given moment, it is packed part of the items represented by $R[i]$, then the vector must be reordered.

In any iteration of the algorithm, it tries to pack C different sets creating a new bin. For that, the algorithm keeps a window of C sets. At first, the window goes from $R[1]$ to $R[C]$. If $\sum_{i=1}^{C} R[i] \geq B$ then the algorithm packs the corresponding sets of $R[1], R[2], \ldots, R[j]$, where $j \leq C$ is the first index such that $\sum_{i=1}^{j} R[i] \geq B$. Notice that $R[j]$ may be partially packed. The totally packed sets are removed from the vector. If $\sum_{i=1}^{C} R[i] < B$ then the algorithm moves the window to the right, until that for the first time the window has C sets such that their sizes are greater than or equal to B. If this is the case, the C sets are packed and the vector R is reordered (if the last considered set was partially packed). Then the algorithm restarts. If in some iteration, the window reaches the end of the vector R, i.e, the C largest sets have total size smaller than B, then the algorithm generates bins by packing entirely C sets in each bin, with exception perhaps in the last bin that can pack less than C sets.

Let $B_1, \ldots, B_N$ be the bins created by the algorithm MW in the order they were created.

Let N_F be the number of full bins and N_C be the number of bins that are not full which we call colored. Let $N = N_F + N_C$. Notice that bins $B_1, \ldots, B_{N_F}$, are the full bins since when the algorithm creates the first non-full bin, when the window reaches the end of R and the C largest sets have total size smaller than B, then all other generated bins becomes non-full having C different sets each except perhaps the last.

Lemma 6.3.1 *If any of the first N_F bins produced by the algorithm* MW *packs less than C different sets (classes), then the algorithm produces an optimal solution.*

Proof. Let B_i be the first bin, among the first N_F bins, that packs less than C different sets. In this case, the window must start from $R[1]$ and goes until $R[j']$ for some $j' \leq C-1$. The vector R is ordered such that $R[j'] \leq R[j'+1] \leq \ldots \leq R[Q]$. Therefore, any $C-1$ remaining sets have total size greater than B. That is, even if the set $R[j']$ was partially packed, all other created bins must be full, because the remaining items of a partially packed set with $C-1$ sets have total size greater than B. □

This way, we consider that for each of the N_F first bins, the algorithm packs, in each iteration, exactly C different sets and that at most one of these sets is partially packed. Clearly, for the remaining N_C bins, all of them packs totally C different sets except perhaps the last bin.

Let $\mathrm{OPT}(I)$ be the number of bins used by an optimal solution to pack instance I. We assume that $N_F \leq \mathrm{OPT}(I) - 1$, otherwise the algorithm generated an optimal solution. We have the following result.

Lemma 6.3.2 *After the* MW *algorithm has created the first $\mathrm{OPT}(I)$ bins, there exists at most N_F sets to be packed.*

Proof. Notice that the number of different sets must satisfy $Q \leq \mathrm{OPT}(I)C$. Since each one of the full bins packs C different sets, where one of these sets may be partially packed, then the algorithm partially packs at most N_F sets. These partially packed sets can be seen as new sets that are considered by the algorithm during its execution. That is, we can assume that the algorithm packs at most $Q + N_F$ different sets. Also remember that each one of the N_C colored bins packs entirely C different sets. Since each one of the first $\mathrm{OPT}(I)$ bins packs C different sets and $Q \leq \mathrm{OPT}(I)C$ we conclude that it remains at most N_F sets that are packed in extra colored bins. □

With this result we can give the approximation factor of the MW algorithm.

Theorem 6.3.3 *The* MW *algorithm has an approximation factor of $(1 + \frac{1}{C})$ for the* CCBP problem when all items are equal sized.

Proof. Let I be an instance for the CCBP problem where all items have unit size. From Lemma 6.3.2, after the algorithm has generated the first $\mathrm{OPT}(I)$ bins, it remains at most N_F sets to be

packed. Since each one of the generated bins packing these sets is colored, each bin entirely packs C different sets and then, the number of extra bins created can be bounded by

$$\left\lceil \frac{N_F}{C} \right\rceil \leq \frac{\mathrm{OPT}(I) - 1}{C} + 1 = \frac{\mathrm{OPT}(I)}{C} - 1/C + 1.$$

We can bound the number of generated bins by $\mathrm{OPT}(I) + \mathrm{OPT}(I)/C + 1$. □

Proposition 6.3.1 *The bound of Theorem 6.3.3 is tight.*

Proof. Consider that the input instance I consists of $N(C-2)$ big sets with $2p+2$ items each, and $2N$ small sets with p items each. The bin capacity is $B = (C-2)(2p+2) + 2p + 2$ items. Notice that $(C-2)$ big sets with two small sets does not fill the bin capacity. When the MW algorithm is executed over this instance, the first generated bin packs one small set, $(C-2)$ big sets entirely and another big set partially. The remaining items of the last packed big set becomes a small set with p items. Notice that the MW algorithm generates $N(C-2)/(C-1)$ bins by packing big sets and one small set that is a residual part of a big set. After that, remains $2N$ small sets that are packed in more $2N/C$ bins. When N and C increase enough, the number of bins tends to $N + N/C$. An optimal pack of this instance uses N bins. In this packing, each bin packs $(C-2)$ big sets and two small sets. □

Notice that the MW algorithm is based in a heuristic that tries to pack C different sets in each bin. But the way the algorithm works, it tends to pack small and large sets in different bins. A good heuristic is to pack large and small sets together, in such a way that each generated bin has a good use of its capacity, while trying to pack C different sets in each bin. For that, we propose a new algorithm that we call Modified-Moving-Window (MW').

Modified-Moving-Window (MW'): This algorithm is similar to the MW algorithm in such a way that it also keeps a window of size C over a vector $R = (R[1], R[2], \ldots, R[Q])$ that is maintained ordered in non-decreasing order of the values $R[i]$. The algorithm also moves a window of size C until the total size of the sets in the window contains B or more items. In the MW' algorithm, we consider that the vector R is a circular list. At first, the window consists of the sets $R[1], \ldots, R[C]$. If the total size of these sets is greater than or equal to B, then the algorithm packs the sets $R[1], \ldots, R[j]$, where $j \leq C$ is the first index such that $\sum_{i=1}^{j} R[i] \geq B$, with the last set $R[j]$ probably partially packed. If the total size of these sets is smaller than B then instead of doing a move to the right, as in the original MW algorithm, the algorithm performs a move to the left and considers the sets $R[Q], R[1], \ldots, R[C-1]$. The algorithm performs moves to the left until the total size of the C sets are greater than or equal to B. In this case it packs the C sets and restarts. If the algorithm performs C moves to the left, and then considers the largest C sets, and this sets have total size less than B, then the algorithm generates a packing like the original MW algorithm, by packing entirely C sets in each bin.

It is not hard to prove similar results to Lemma 6.3.1 and Lemma 6.3.2 to the MW' algorithm. Using the same arguments of Theorem 6.3.3 we can prove the following result.

Theorem 6.3.4 *The* MW' *algorithm has an approximation factor of* $(1 + \frac{1}{C})$ *for the* CCBP problem where all items are equal sized.

Notice that this bound is tight since the algorithm MW' generates the same solution generated by the algorithm MW for the instance presented in Proposition 6.3.1. The advantage of the MW' algorithm is to try to pack small sets with large ones trying to guarantee a good filling of the bins, since it tries to pack the maximum number of small sets with large sets. To see this, consider for example an instance I that consists of $2n$ small sets, each one with one item, n large sets with 5 items each and n medium sets with 2 items each. Suppose $B = 7$ and $C = 3$. The MW algorithm first generates n bins by packing two medium sets and part of another large set. After that, it generates $2n/3$ new bins to pack the small sets. The MW' algorithm first generates n bins such that each one packs two small sets and a large set. The remaining medium sets are packed in $n/3$ bins.

Another simple approach used to solve the problem is to use similar ideas of the well known *FFD*, (*BFD*) algorithms (see Coffman *et al.* [2])

Algorithm FFD**:** The algorithm first sorts the sets $U_1, \ldots, U_M$ in non-increasing order of their size and then apply the FF algorithm in the list obtained concatenating these sets.

Theorem 6.3.5 *The* FFD *algorithm has an approximation factor equal to 2 for the* CCBP *problem when all items have unit size.*

Proof. Let $B_1, \ldots, B_N$ be the bins created by the algorithm, N_F be the number of full bins and N_C be the number of colored bins. Clearly, $N_F \leq \mathrm{OPT}$ and each bin that is not full must be colored except perhaps the last generated bin. Also notice that two different bins that are colored cannot have items of a same color. Since $CN_C/C \leq \lceil Q/C \rceil \leq \mathrm{OPT}$ we get that $N_C \leq \mathrm{OPT}$. Then we can bound the number of generated bins by $N \leq 2\mathrm{OPT} + 1$. □

Since this algorithm does not try to optimize the class usage in the packing, it can generate poor quality packings. In fact, we show in the next proposition that the bound of Theorem 6.3.5 is tight.

Proposition 6.3.2 *The bound of Theorem 6.3.5 is tight.*

Proof. Let $I = (L, s, c, C, Q)$ be an instance to the CCBP problem where all items have unit size. Let the size of the bins be $B = C^2$. Suppose the input list of items consists of one big set with C^3 items and C^2 small sets with one item each. The FFD algorithm first packs the big set in C^3/C^2 bins and the small sets in C^2/C bins giving a total of $2C$ bins. An optimal solution uses C bins packing in each bin $C^2 - (C - 1)$ items of the big set and $C - 1$ small sets. The remaining $C(C - 1)$ items of the big set, and C small sets can be packed in 2 extra bins. □

Now we consider the case where items in each set may have different sizes. This case is also interesting for applications of the data-placement problem to video-on-demand servers.

Suppose that users have different network access speeds. In this case, requests for load resources may have different sizes. This case can be mapped to the case in the CCBP problem where items have different sizes. Also notice that even if the items have different sizes, in practical instances it is expected that the size of the item is not too large. So, suppose that the maximum size of an item is an integer bounded by B/m for some $m \geq 1$. Problems with this restriction are also called parametric packing problems [16, 3]. Given an integer m, we denote this version of the problem as Parametric Class Constrained Bin Packing ($\mathrm{CCBP_m}$) problem.

Let I be an instance of the $\mathrm{CCBP_m}$ problem where each item has size bounded by B/m. Consider that the input instance I consists of sets $U_1, \ldots, U_Q$. We now present an algorithm to pack this instance. Although items may have different sizes, consider that each item with size s greater than 1 is broken into s unit size pieces. Now apply the MW algorithm for this modified instance. Now consider this packing for the original items. For each full bin it may happen that the last item packed is fractionally packed. For each bin where this happens, remove the item of the bin. Notice that there are at most N_F items removed of the generated packing. For these remaining items, generate new bins packing at least $\min\{m, C\}$ items in each bin except perhaps in the last bin.

Theorem 6.3.6 *There exists an algorithm for the* CCBP problem where each item has size at most B/m, for some $m \geq 1$, with approximation factor equal to $(1 + 1/\min\{m, C\} + 1/C)$.

Proof. From Theorem 6.3.3, the packing generated when items are fractionally packed uses at most $(1 + 1/C)\mathrm{OPT}(I) + 1$ bins. Notice that the number of items fractionally packed in this packing is bounded by N_F, since the first N_F bins are the only ones that are full. These N_F extra items can be packed in at most $\lceil N_F/\min\{m, C\}\rceil$ extra bins. □

6.4 The Online CCBP Problem

From now on, we consider that the capacity of the bin is $B = 1$, and each item e has size $0 < s_e \leq 1$. In this section we consider the online class constrained bin packing problem. In this case each item in the list of items $L = (a_1, \ldots, a_n)$, is packed without knowledge of subsequent items in the list. In subsection 6.4.1 we present lower bounds for any bounded space algorithm, in subsection 6.4.2 we present and analyze an algorithm based in the First-Fit strategy and finally in subsection 6.4.3 we present another online algorithm with better competitive ratio.

6.4.1 Lower bounds for bounded space algorithms

In this section we present inapproximability results for the bounded space online CCBP problem. In this case, the basic strategy is to compare the result obtained by the algorithm with the

optimum offline packing.

Theorem 6.4.1 *Let l be a constant, then the l-bounded space online* CCBP *problem does not admit an algorithm with constant competitive ratio.*

Proof. Let $\mathcal{A}$ be an algorithm for the l-bounded space online CCBP problem. Consider an instance I, such that $|L| = n^2 l$, $Q = nl$, and n is divisible by C. The list L have nl different classes and all items have size $1/Cn$. Consider that $L = L_1 \| \ldots \| L_n$, where each $L_i = (a_1, \ldots, a_{nl})$ is a sequence of nl items where each a_j has class j.

Let t_i be the time immediately after the algorithm has packed the list L_i. At time t_1 the algorithm $\mathcal{A}$ can have at most l open bins. Since each item of the first sequence is of a different class, the algorithm uses at least nl/C bins to pack L_1, where at least $nl/C - l$ of these bins are closed. When the packing of the list L_2 starts, the algorithm has at most l open bins that can pack at most lC items of the sequence L_2. To pack this sequence, the algorithm uses at least $(ln - lC)/C$ bins. This is also valid for the other sequences $L_3, \ldots, L_n$.

Therefore, to pack the list L, the algorithm $\mathcal{A}$ uses at least

$$n(nl/C) - (n-1)l = n^2 l/C - (n-1)l$$

bins.

Since all items have size $1/Cn$, an optimal offline solution can use at most ln/C bins, by packing Cn items in each bin. Therefore, the competitive ratio must be at least

$$\lim_{n\to\infty} \frac{n^2 l/C - (n-1)l}{nl/C} = n.$$

□

In Theorem 6.4.1, items may have arbitrary small sizes. If all items have size at least ε, for some constant ε, we may also obtain an inapproximability result using similar arguments. Notice that in this case, any simple algorithm has a competitive ratio of $1/\varepsilon$.

Theorem 6.4.2 *Let l and $\varepsilon < 1$ be constants and consider instances for the* CCBP problem where each item has size at least ε. Then the online CCBP problem does not admit an algorithm with competitive ratio better than $O(1/C\varepsilon)$.

Proof. Suppose that $1/\varepsilon$ divides n and we have the same instance presented in Theorem 6.4.1, modified such that all items have size equal to ε. In this case any algorithm uses at least $n^2 l/C - (n-1)l$ bins. An optimal offline solution packs items of a given class in $n\varepsilon$ bins. To pack L an optimal offline algorithm uses at most $n^2 l\varepsilon$ bins.

Therefore, the competitive ratio is at least

$$\lim_{n\to\infty} \frac{n^2 l/C}{n^2 l\varepsilon} - \frac{nl - l}{n^2 l\varepsilon} = \frac{1}{C\varepsilon}.$$

□

Given these negative results, for the remaining of this section we only consider the unbounded space online CCBP problem.

6.4.2 The First-Fit Algorithm

Given an online algorithm $\mathcal{A}$ for the bin-packing problem, we can obtain an online algorithm $\mathcal{A}^*$ for the online CCBP problem in a straightforward manner. To pack the next item e, the algorithm $\mathcal{A}^*$ works as follows: Let c_e be the class of the item e, $\mathcal{B}$ be the list of bins in the order they were opened. Let $\mathcal{B}_e$ be the list of bins of $\mathcal{B}$, in the same order of $\mathcal{B}$, where each bin has at least one item of class c_e or has items of at most $C-1$ different classes. The item e is packed with algorithm $\mathcal{A}$ into the bins of $\mathcal{B}_e$.

One of the most famous algorithm for the bin-packing problem is the First-Fit (FF) algorithm. This algorithm packs the next item into the first bin, in the order they were opened, that has sufficient space for the item.

In this section we show that the competitive ratio of the algorithm FF* is in $[2.7, 3]$. We note that the upper bound was previously shown by Dawande *et al.* [4]. Notice that the algorithm FF* is online, since it only looks for the item it is packing and it is unbounded since it keeps all bins opened. In fact it closes a bin only if the bin is full. This algorithm is used in other algorithms of subsequent sections.

Lemma 6.4.3 *Let I be an instance for the online* CCBP *problem such that every item has size at most ε. Let $\mathcal{P}$ be the set of bins generated by the algorithm* FF**, applied over the instance I, that are filled by less than $1-\varepsilon$. Then: (i) Each bin in $\mathcal{P}$, which is not the last generated bin, is colored. (ii) There is no items of a same color in two different bins of $\mathcal{P}$.*

Proof. Let B_1 be a bin in $\mathcal{P}$, B_l the last bin created by the algorithm FF* and a_l an item packed in B_l. Since B_1 is filled with less than $1-\varepsilon$ and $s(a_l) \leq \varepsilon$, a_l was not packed in B_1 because it must be colored.

Now suppose there are two different bins B_1 and B_2 in $\mathcal{P}$ that are filled with less than $1-\varepsilon$ and there are items $a_i \in B_i$, $i = 1, 2$ with the same class. Without loss of generality, consider that B_1 was opened first. Since the maximum size of a_2 is ε and the algorithm FF* tries to pack an item into the bins in the order they were opened, satisfying the size and class constraints, the item a_2 would be packed in the bin B_1. That is, a contradiction. □

The result of the next theorem can be found in the work of Dawande *et al.* [4]. The idea to prove this theorem is to consider separately bins that are filled by at least half of its capacity and bins that are not. In the first case the number of bins is bounded by $2\mathrm{OPT}(I)$. In the later case using Lemma 6.4.3 we can prove that all bins are colored, except perhaps the last, and then using the fact that $\lceil Q/C \rceil \leq \mathrm{OPT}(I)$, we can bound the number of used bins by $\mathrm{OPT}(I)+1$.

Theorem 6.4.4 *The algorithm* FF* *has a competitive ratio* 3 *for the online* CCBP *problem.*

Now, we show that the algorithm FF* cannot have a competitive ratio better than 2.7. We first give an intuitive lower bound of 2.666 and then we present the lower bound of 2.7.

Theorem 6.4.5 *There is an instance* I_n *with* n *items,* $n \geq 1$*, for the online* CCBP *problem such that* $\text{FF}^*(I_n)/\text{OPT}(I_n) \to 2.666$ *as* $n \to \infty$.

Proof. Let I be an instance with an input list of items $L = L_a \| L_b \| L_c \| L_d$. Let C be the maximum number of classes allowable in each bin. The list $L_a = (a_1, \ldots, a_{(C-1)6N})$ is such that each item a_i has class i, $i = 1, \ldots, (C-1)6N$ and each item has size α, which is a very small value. This list is followed by a list $L_b = (b_1, \ldots, b_{6N})$, where each item b_i has class $r = 6N(C-1)+1$, and size $1/7 + \varepsilon$. In the list $L_c = (c_1, \ldots, c_{6N})$ each item c_i has size $1/3 + \varepsilon$ and class r. Finally, in the list $L_d = (d_1, \ldots, d_{6N})$ each item d_i has size $1/2 + \varepsilon$ and class r.

The FF* algorithm packs the list L_a in $\frac{6N(C-1)}{C}$ bins, the list L_b in N bins, the list L_c in $3N$ bins and the list L_d in $6N$ bins. The Figure 6.2 presents the different bins in the packing generated by the FF* algorithm.

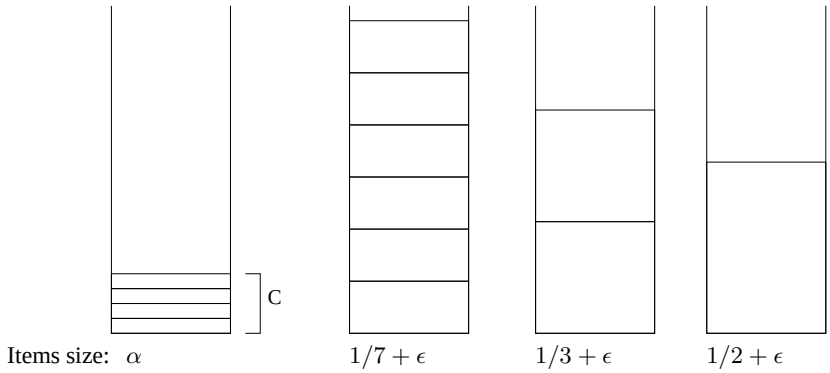

Figure 6.2: The bins generated by the FF* algorithm.

An optimal (offline) solution uses at most $6N$ bins. This packing is obtained by packing one item of L_d, one item of L_c, one item of L_b and $C-1$ items of the list L_a in only one bin.

This gives a lower bound of

$$\lim_{N,C\to\infty} \frac{\frac{(C-1)6N}{C} + 10N}{6N} = 2.666.$$

□

The previous lower bound can be improved using an intricate instance presented by Johnson *et al.* [11] that provides a lower bound of 1.7 for the FF algorithm in the bin packing problem.

Theorem 6.4.6 *The competitive ratio of the algorithm* FF* *is at least* 2.7.

Proof. Consider an instance I such that each bin can pack at most C different classes. The input list L is the concatenation of four lists: $L = L_a \| L_b \| L_c \| L_d$. In the list $L_a = (a_1, \ldots, a_{5N(C-1)})$, each item a_i has class i, for $i = 1, \ldots, 5N(C-1)$, and each item has size α, which is a very small value. The list L_a is followed by an instance similar to the one presented by Johnson *et al.* [11] that provides a lower bound of 1.7 for the FF algorithm in the bin packing problem. In the list $L_b = (b_1, \ldots, b_{5N})$ each item b_i has size $1/7 + y_i$, where $y_i \in \mathcal{R}$, for $i = 1, \ldots, 5N$. In the list $L_c = (c_1, \ldots, c_{5N})$ each item c_i has size $1/3 + w_i$, where $w_i \in \mathcal{R}$, for $i = 1, \ldots, 5N$. In the list $L_d = (d_1, \ldots, d_{5N})$ each item d_i has size $1/2 + \varepsilon$. All items in the lists L_b, L_c and L_d have class $5N(C-1)+1$.

The algorithm FF* generates a packing as the one presented in the proof of the Theorem 6.4.5, except that it packs only five items of the list L_b per bin. That is,

$$\text{FF}^*(I) \geq \frac{5N(C-1)}{C} + N + 2.5N + 5N.$$

An optimal solution can use $5N+2$ bins (see [11]), packing one item of each list L_b, L_c and L_d and $C-1$ items of the list L_a.

Therefore, the competitive ratio of the algorithm FF* is at least

$$\lim_{N,C\to\infty} \frac{5N(C-1)/C + 8.5N}{5N+2} = 2.7.$$

□

6.4.3 A 2.75-competitive algorithm

In this section we present an algorithm, which we denote by $\mathcal{A}_C$ (Figure 6.3), with competitive ratio in the interval $(2.666, 2.75]$

To prove the competitive ratio of the algorithm $\mathcal{A}_C$, we use the following lemma (the proof can be found in [16]).

Lemma 6.4.7 *Suppose* X, Y, x, y *are real numbers such that* $x > 0$ *and* $0 < X < Y < 1$. *Then*

$$\frac{x+y}{\max\{x,\ X\,x + Y\,y\}} \leq 1 + \frac{1-X}{Y}.$$

Theorem 6.4.8 *Algorithm* $\mathcal{A}_C$ *has a competitive ratio of* 2.75.

Proof.

Let L_i the list of items packed in $\mathcal{P}_i$, for $i = 1, 2, 3$.

ALGORITHM $\mathcal{A}_C(L, s, c, C, Q)$

1. Let $\mathcal{P}_i \leftarrow \emptyset$, for $i = 1, 2, 3$.
2. For each $e \in L$ do
3. if $s(e) \in (\frac{1}{2}, 1]$ then $k \leftarrow 1$.
4. if $s(e) \in (\frac{1}{3}, \frac{1}{2}]$ then $k \leftarrow 2$.
5. if $s(e) \in (0, \frac{1}{3}]$ then $k \leftarrow 3$.
6. Let $\mathcal{P}'_k$ the sublist of bins in $\mathcal{P}_k$ having items of class $c(e)$ or with at most $C - 1$ classes, preserving the order of the bins in $\mathcal{P}_k$.
7. If possible pack the item e into the bins $\mathcal{P}'_k$ using the algorithm FF*. Otherwise, pack e into a new empty bin in $\mathcal{P}_k$.
8. Return $\mathcal{P}_1 \| \mathcal{P}_2 \| \mathcal{P}_3$.

Figure 6.3: Algorithm $\mathcal{A}_C$.

Note that all bins of $\mathcal{P}_1$ have exactly one item with size greater than $\frac{1}{2}$. In fact we cannot pack more than one item of L_1 per bin. Therefore,

$$|\mathcal{P}_1| \leq \mathrm{OPT}(I) \tag{6.1}$$

$$\frac{1}{2}|\mathcal{P}_1| \leq s(L_1). \tag{6.2}$$

The packing $\mathcal{P}_2$ has exactly two items per bin, except perhaps the last, each item with size at least $\frac{1}{3}$. Therefore,

$$(|\mathcal{P}_2| - 1)\frac{2}{3} \leq s(L_2). \tag{6.3}$$

Let $\mathcal{P}'_3$ the set of bins in $\mathcal{P}_3$ that are filled by at least $\frac{2}{3}$ and $\mathcal{P}''_3$ the remaining bins (i.e., $\mathcal{P}''_3 = \mathcal{P}_3 \setminus \mathcal{P}'_3$). The following is valid

$$(|\mathcal{P}'_3|)\frac{2}{3} \leq s(L'_3). \tag{6.4}$$

where L'_3 is the set of items packed in $\mathcal{P}'_3$. Let $N_A = |\mathcal{P}_1|$ and $N_B = |\mathcal{P}_2| + |\mathcal{P}'_3| - 1$. Since $\mathrm{OPT}(I) \geq s(I) \geq s(L_1) + s(L_2 \| L'_3)$ from inequalities (6.2)–(6.4) we have

$$\begin{aligned} \mathrm{OPT}(I) &\geq s(I) \geq s(L_1) + s(L_2 \| L'_3) \\ &\geq \frac{1}{2}N_A + \frac{2}{3}N_B. \end{aligned} \tag{6.5}$$

From inequalities (6.1) and (6.5) we have

$$\mathrm{OPT}(I) \geq \max\{N_A, \frac{1}{2}N_A + \frac{2}{3}N_B\}. \tag{6.6}$$

From Lemma 6.4.7 we have that

$$|\mathcal{P}_1| + |\mathcal{P}_2| + |\mathcal{P}'_3| \leq \frac{N_A + N_B}{\max\{N_A, \frac{1}{2}N_A + \frac{2}{3}N_B\}}\mathrm{OPT}(I) + 1 \quad (6.7)$$

$$\leq 1.75\,\mathrm{OPT}(I) + 1. \quad (6.8)$$

Now, consider the packing $\mathcal{P}''_3$. Using a similar argument used in Lemma 6.4.3, we have

$$|\mathcal{P}''_3| - 1 \leq \frac{Q}{C} \leq \mathrm{OPT}(I). \quad (6.9)$$

The proof can be completed summing the inequalities (6.8) and (6.9).

$$\begin{aligned}\mathcal{A}_C(I) &= |\mathcal{P}_1| + |\mathcal{P}_2| + |\mathcal{P}'_3| + |\mathcal{P}''_3| \\ &\leq 1.75\,\mathrm{OPT}(I) + \mathrm{OPT}(I) + 2 = 2.75\,\mathrm{OPT}(I) + 2.\end{aligned}$$

□

Notice that the same instance used to prove a lower bound for the algorithm FF* in Theorem 6.4.5 can be used to prove a lower bound for the $\mathcal{A}_C$ algorithm.

Theorem 6.4.9 *There is an instance I for the online* CCBP *problem such that* $\mathcal{A}_C(I)/\mathrm{OPT}(I) \geq 2.666$.

6.5 An APTAS for Bounded Number of Classes

In this section we present an APTAS for the offline VCCBP problem. The input instance for this problem is a tuple $I = (L, s, c, w, C, Q)$ where $w : \{1, \ldots, T\} \to \mathbb{R}^+$ is a function of bins size. The problem is to find a pack of all items minimizing the total size of used bins. In this section we consider that the maximum size of a bin is 1 and that the number of different classes Q in the input instance, is bounded by a constant.

In subsection 6.5.1 we present the algorithm of Dawande, Kalagnanam and Sethuraman [5, 4] and show in what points their algorithm failed to be an APTAS. In subsection 6.5.2 we present an APTAS for the VCCBP problem. Given an ε, we will show an algorithm $\mathcal{A}$ that runs in polynomial time and produces a packing for a given instance such that $\mathcal{A}(I) \leq (1 - O(\varepsilon))\mathrm{OPT} + \beta$, where β is a constant.

As was noticed by Dawande *et al.* [5, 4], we only use bins such that their size are at least ε, since this condition does not affect too much the cost of the solution, i.e, the algorithm remains an APTAS.

6.5.1 The Algorithm of Dawande, Kalagnanam and Sethuraman

In this section we give a brief description of the algorithm of Dawande *et al.* [5, 4] and present the points where their algorithm fails.

Let $I = (L, s, c, w, C, Q)$ be an instance for the VCCBP problem and let L_b be the items in L with size at least ε^2 (big items) and let L_s be the remaining items in L (small items).

Let $n = |L_b|$. The algorithm sorts the list L_b in non-increasing order of size and partition this list into groups (lists) $L_1, \ldots, L_M$, each one with $\lceil n\varepsilon^2 \rceil$ items except perhaps the last list that can has less than $\lceil n\varepsilon^2 \rceil$ items. Call the first item in each group as the group-leader. Let L'_i be the list having $|L'_i| = |L_i|$ items, where each item has size equal to the size of the group-leader of L_i. Let $L' = L'_1 \| \ldots \| L'_M$.

For the list L' it is possible to generate all configurations of bins in constant time since the number of different items size is bounded by a constant M, the number of different item colors is also bounded by a constant Q and the maximum number of items that can be packed in a bin is $1/\varepsilon^2$. Let $t = MQ$. Given an item size and an item color, denote by d_i the number of items of this type $i \in [t]$.

Let N be the total number of bin configurations. Let x_j be a variable that represents the number of times a configuration $j \in [N]$ is used in a solution, a_{ij} be the coefficient that represents the number of times an item type $i \in [t]$ is used in configuration j and w_j the size of the bin used in configuration j. The next step of the algorithm is to solve the following linear program:

$$
\begin{aligned}
& \min \sum_{j=1}^{N} w_j x_j & & \\
\sum_{j=1}^{N} a_{ij} x_j & \geq d_i & \forall\, i \in [t] \quad & (1) \\
x_j & \geq 0 & \forall\, j \in [N]. \quad & (2)
\end{aligned}
\qquad \text{(LP)}
$$

The algorithm solves this linear program and generates an integer solution by rounding up the variables x. The solution is a packing for the list L' that is used to generate a packing for the list L_b.

The next step of the algorithm is to pack the small items in the solution provided by the linear program. To do this, it uses the FF* algorithm.

Dawande *et al.* [5, 4] claimed that this algorithm is an APTAS for the VCCBP problem.

The list L_b was partitioned into lists $L_1 \| \ldots \| L_M$. Let L''_i be a list having $|L''_i| = |L_i|$ items, where each item has size equal to the group-leader of the list L_{i+1}, for $i = 1, \ldots, M-1$, and L''_M be an empty list. Let $L'' = L''_1 \| \ldots \| L''_M$. Clearly $\mathrm{OPT}(L'') \leq \mathrm{OPT}(L_b)$.

Dawande *et al.* claimed that the following relation is valid

$$OPT(L') \leq OPT(L'') + \lceil n\varepsilon^2 \rceil \leq OPT(L_b) + \lceil n\varepsilon^2 \rceil,$$

given the argument that L' and L'' differ only in their first and last groups. This way, given a packing for the list L'' it is easy to construct a packing for the list $L'_2 \| \ldots \| L'_M$. Since $|L'_i| = |L''_{i-1}|$, for $i = 2, \ldots, M$, and their items size are the same, although this seems to be true, notice that the color of items of L'_i and L''_{i-1} may be different. Then, it is not clear how to construct a packing for $L'_2 \| \ldots \| L'_M$ given a packing for L''.

Let B be the number of bins used by their algorithm. After packing the small items using the first-fit strategy, they claimed that at least $B - \lceil \frac{Q}{C} \rceil$ bins have residual capacity at most ε. This is also not true. Suppose all small items have different colors from the big items. It is easy to construct examples where optimal packings for the big items given by the linear program have all bins with C different colors and the residual space is larger than a given ε. This way no small item will be packed in the bins given as solution by the linear program and then, all these bins will have residual capacity greater than ϵ.

6.5.2 An APTAS for the VCCBP Problem

In this section we present an APTAS for the VCCBP problem. In the next subsection we show how to pack big items doing a linear rounding for each different color. The algorithm to pack the big items generates a polynomial number of packings for the big items, and also provide information of how to pack small items. In the following subsection, we present an algorithm to pack the small items that is based in the solution of a linear program. The algorithm generates a polynomial number of packings such that at least one is very close to the optimal.

Packing Big Items with Linear Rounding

Let L_b be the items in L with size at least ε^2 (big items) and let L_s be the remaining items in L (small items). In this section we show how to do the linear rounding for the big items and generate a packing for them.

The algorithm that packs the list L_b, which we denote by $\mathrm{A_{LR}}$, uses the linear rounding technique, presented by Fernandez de la Vega and Lueker [6], and considers only items with size at least ε^2. The algorithm $\mathrm{A_{LR}}$ returns a pair $(\mathcal{P}_B, \mathbb{P})$, where $\mathcal{P}_B$ is a packing for a list of very big items and $\mathbb{P}$ is a set of packings for the remaining items of L_b.

For the use of the linear rounding technique, we use the following notation: Given two lists of items X and Y, let $X_1, \ldots, X_Q$ and $Y_1, \ldots, Y_Q$ be the partition of X and Y respectively in colors, where X_c and Y_c have only items of color c for each $c \in [Q]$. We write $X \preceq Y$ if there is an injection $f_c : X_c \to Y_c$ for each $c \in [Q]$ such that $s(e) \leq s(f(e))$ for all $e \in X_c$.

For any instance X, denote by $\overline{X}$ the instance with precisely $|X|$ items with size equal to the size of the smallest item in X. Clearly, $\overline{X} \preceq X$.

The Algorithm also uses the variant of the First-Fit (FF*) that we presented in section 6.4.2.

The algorithm $\mathrm{A_{LR}}$ is presented in Figure 6.4. It consists in the following: Let $L_1, \ldots, L_Q$ be the partition of the input list L_b into colors $1, \ldots, Q$ and let $n_c = |L_c|$ for each color c. The algorithm $\mathrm{A_{LR}}$ sorts each list L_c in non-increasing order of size and then partition the list L_c into at most $M = \lceil 1/\varepsilon^3 \rceil$ groups $L_c^1, L_c^2, \ldots, L_c^M$, where $L_c = L_c^1 \| \ldots \| L_c^M$. Each group has $\lfloor n_c\varepsilon^3 \rfloor$ items except perhaps the last list (with the smallest items) that can have less than $\lfloor n_c\varepsilon^3 \rfloor$ items.

Let $L_B = \cup_{c=1}^{Q} L_c^1$. The algorithm generates a packing $\mathcal{P}_B$ of the list L_B with cost at most $O(\varepsilon)\mathrm{OPT}(I)$ and a set $\mathbb{P}$ with a polynomial number of packings for the items in $L_b \setminus L_B$. The packing $\mathcal{P}_B$ is generated by the algorithm FF* with bins of size 1.

The algorithm generates a set of packings $\mathbb{Q}$, of polynomial size, for the list $(\overline{L_1^1} \| \ldots \| \overline{L_1^{M-1}} \| \ldots \| \overline{L_Q^1} \| \ldots \| \overline{L_Q^{M-1}})$. This can be done in polynomial time as the next lemma guarantees.

Lemma 6.5.1 *Given an instance $I = (L_b, s, c, w, C, Q)$, where the number of distinct items sizes of each color is at most a constant M, the number of different colors is bounded by a constant Q and each item $e \in L_b$ has size $s_e \geq \varepsilon^2$, then there exists a polynomial time algorithm that generates all possible packings of L_b. Moreover, each bin of each generated packing has an indication of the possible colors that may be used by further small items.*

Proof. The number of items in a bin is bounded by $y = 1/\varepsilon^2$. The number of distinct type of items is bounded by MQ. The number of different configurations of bins is bounded by $r' = \binom{y+MQ+1}{y}$. If we want to indicate the colors of small items that should be packed in each configuration, the number of different configurations will be $r = r'2^Q$, which is a constant. Notice that we only generate configurations that satisfy the color constraints.

For each given configuration, we pack it with the smallest bin that has enough space to pack the configuration. The number of all feasible packings is bounded by $\binom{n+r}{n}$, which is bounded by $(n+r)^r$, which in turn is polynomial in n. ∎

Since $\overline{L_c^i} \succeq L_c^{i+1},\ i = 1, \ldots, M-1$ for each color c, it is easy to construct a packing for the list $L_1^2 \| \ldots \| L_1^M \| \ldots \| L_Q^2 \| \ldots \| L_Q^M$, given a packing for the list

$$(\overline{L_1^1} \| \ldots \| \overline{L_1^{M-1}} \| \ldots \| \overline{L_Q^1} \| \ldots \| \overline{L_Q^{M-1}}).$$

The following is valid for the packing $\mathcal{P}_B$ of the list L_B.

Lemma 6.5.2 $w(\mathcal{P}_B) \leq Q\varepsilon\mathrm{OPT}(I)$.

Proof. Notice that the algorithm FF* packs at least one item per bin and since $|L_B| \leq Qn\varepsilon^3$ and each item has size at least ε^2, we have $|L_B| \leq Q\varepsilon\mathrm{OPT}(I)$.

∎

ALGORITHM $\mathrm{A_{LR}}(L_b)$

Input: List L_b with n items, each item $e \in L_b$ with size $s_e \geq \varepsilon^2$.

Output: A pair $(\mathcal{P}_B, \mathbb{P})$, where $\mathcal{P}_B$ is a packing and $\mathbb{P}$ is a set of packings, where $\mathcal{P}_B \cup \mathcal{P}'$ is a packing of L_b for each $\mathcal{P}' \in \mathbb{P}$.

1. Partition L_b into lists L_c for each color $c = 1, \ldots, Q$ and let $n_c = |L_c|$.

2. Sort each list L_c in non-increasing order of items size.

3. Partition each list L_c into $M \leq \lceil 1/\varepsilon^3 \rceil$ groups $L_c^1, L_c^2, \ldots, L_c^M$, such that

$$\overline{L_c^i} \succeq L_c^{i+1},\ i = 1, \ldots, M-1$$
$$\text{where } |L_c^i| = q_c = \lfloor n_c \varepsilon^3 \rfloor \text{ for all } i = 1, \ldots, M-1,$$
$$\text{and } |L_c^M| \leq q_c.$$

4. Let $L_B = \cup_{c=1}^{Q} L_c^1$.

5. Let $\mathcal{P}_B$ be a packing of L_B obtained by the algorithm FF* with bins of size 1.

6. Let $\mathbb{Q}$ be the set of all possible packings over the list $(\overline{L_1^1} \| \ldots \| \overline{L_1^{M-1}} \| \ldots \| \overline{L_Q^1} \| \ldots \| \overline{L_Q^{M-1}})$, according to Lemma 6.5.1.

7. Let $\mathbb{P}$ be the set of packings for the items in $(L_1^2 \| \ldots \| L_1^M \| \ldots \| L_Q^2 \| \ldots \| L_Q^M)$, using the packings $\mathcal{Q} \in \mathbb{Q}$.

8. Return $(\mathcal{P}_B, \mathbb{P})$.

Figure 6.4: Algorithm to obtain packings for items with size at least ε^2.

Packing the small items

Observe that algorithm $\mathrm{A_{LR}}$ generates a packing for very big items that costs at most $Q\varepsilon\mathrm{OPT}(I)$, and a set $\mathbb{P}$ of packings for the remaining big items. For a given packing $\mathcal{P} \in \mathbb{P}$, the algorithm marked colors of small items that should be packed in each bin of $\mathcal{P}$. To pack the small items we use a solution given by a linear program.

Let $\mathcal{P} = \{B_1, \ldots, B_k\}$ be a packing of the list of items L_b and suppose we have to pack a list L_s of small items, with size at most ε^2, into $\mathcal{P}$. The packing of the small items is obtained from a solution of a linear program. Let $N_i \subseteq [Q]$ be the set of possible colors that may be used to pack the small items in the bin B_i of the packing $\mathcal{P}$. For each color $c \in N_i$, define a non-negative variable x_c^i. The variable x_c^i indicates the total size of small items of color c to be packed in the bin B_i. Denote by $s(B_i)$ the total size of items already packed in the bin B_i and by $w(B_i)$ the capacity of bin B_i. Consider the following linear program denoted by LPS:

$$\begin{aligned} & \max \sum_{i=1}^{k} \sum_{c \in N_i} x_c^i \\ s(B_i) + \sum_{c \in N_i} x_c^i & \leq w(B_i) \qquad \forall\, i \in [k] \quad (1) \\ \sum_{i=1}^{k} x_c^i & \leq s(S_c) \qquad \forall\, c \in [C], \quad (2) \end{aligned} \qquad \text{(LPS)}$$

where S_c is the set of small items of color c in S.

The constraint (1) guarantees that the items packed in each bin satisfy its capacities and constraint (2) guarantees that variables x_c^i is not greater than the total size of small items.

Given a packing $\mathcal{P}$, and a list L_s of small items, the algorithm first solves the linear program LPS, and then packs small items in the following way: For each variable x_c^i it packs, while possible, the small items of color c into the bin B_i, so that the total size of the packed small items is at most x_c^i. The possible remaining small items are packed using the algorithm FF* into new bins of size 1. The algorithm to pack small items has polynomial time, since the linear program *LPS* can be solved in polynomial time.

The small items that are packed into new bins use at most

$$\left\lceil \frac{(s(L_s) - \sum_{i=1}^{k} \sum_{c \in N_i} x_c^i)}{(1-\varepsilon^2)} + \frac{|\mathcal{P}|\varepsilon^2 Q}{(1-\varepsilon^2)} \right\rceil + \lceil Q/C \rceil$$

new bins, since each bin is filled by at least $(1-\varepsilon^2)$ except perhaps by at most $\lceil Q/C \rceil$ bins.

The algorithm packs the small items in each packing $\mathcal{P} \in \mathbb{P}$. In the end, the algorithm generates another set of packings $\mathbb{P}'$ for all items. At least one of the generated packings has cost at most $(1+O(\varepsilon))\mathrm{OPT}(I) + \beta$, for a constant β. The algorithm returns the packing with smallest cost.

Now we prove that the presented algorithm is an APTAS for the VCCBP.

Theorem 6.5.3 *Let $I = (L, s, c, w, C, Q)$, be an instance for the* VCCBP *problem. The packing $\mathcal{P}$ returned by the algorithm satisfy $w(\mathcal{P}) \leq (1+O(\varepsilon))\mathrm{OPT}(I) + \beta$, where β is a constant.*

Proof. Let O be an optimal packing for instance I. Let O' be the packing without the small items and with the big items rounded according to the linear rounding of algorithm $\mathrm{A_{LR}}$. Assume that each bin of O' has an indication of the colors of small items used in the corresponding bin of O. Clearly the packing $O' \in \mathbb{Q}$ except that it can use smaller bins than the ones used in O.

When the algorithm generates a packing $\mathcal{P}$ for the list $L_1^2 \| \ldots \| L_1^M \| \ldots \| L_Q^2 \| \ldots \| L_Q^M$ using the packing O' with items $(\overline{L_1^1} \| \ldots \| \overline{L_1^{M-1}} \| \ldots \| \overline{L_Q^1} \| \ldots \| \overline{L_Q^{M-1}})$, it is true that $w(\mathcal{P}) \leq w(O)$ since in $\mathcal{P}$ we probably use bins of smaller size for each given configuration of big items.

Let $\mathcal{P} = \{B_1, \ldots, B_k\}$. Notice that we must have

$$w(O) \geq w(\mathcal{P}) + (s(L_s) - \sum_{i=1}^{k} \sum_{c \in N_i} x_c^i).$$

The total size of small items that are packed into new bins is at most

$$(s(L_s) - \sum_{i=1}^{k} \sum_{c \in N_i} x_c^i) + |\mathcal{P}|\varepsilon^2 Q.$$

The algorithm packs small items in bins of size 1 obtaining a new packing $\mathcal{P}'$. The total cost of the packing $\mathcal{P}'$ is

$$\begin{aligned} w(\mathcal{P}') &\leq w(\mathcal{P}) + \lceil \frac{(s(L_s) - \sum_{i=1}^{k} \sum_{c \in N_i} x_c^i)}{(1-\varepsilon^2)} + \frac{|\mathcal{P}|\varepsilon^2 Q}{(1-\varepsilon^2)} \rceil + \lceil Q/C \rceil & (6.10) \\ &\leq \frac{w(O)}{(1-\varepsilon^2)} + \frac{|\mathcal{P}|\varepsilon^2 Q}{(1-\varepsilon^2)} + \lceil Q/C \rceil + 1 & (6.11) \\ &\leq \frac{w(O)}{(1-\varepsilon^2)} + \frac{\varepsilon Q w(O)}{(1-\varepsilon^2)} + \lceil Q/C \rceil + 1. & (6.12) \end{aligned}$$

The last inequality follows from the fact that $|\mathcal{P}| \leq |O|$ and the smallest size of a bin is ε. Using this result, Lemma 6.5.2 and the fact that Q is bounded by a constant we conclude the proof. □

6.6 Experimental Results of the Practical Algorithms

In this section we provide experimental results for the algorithms MW, MW′ and FFD presented in Section 6.3. As we mentioned, these algorithms were developed motivated by the data placement problem in video servers. This problem is a special case of the CCBP problem. All these algorithms were implemented in C and we made a series of practical tests with them.

The instance set is constructed in some way to represent the real problem. A movie in MPEG format uses about 2Gbytes of space, and requires a transference rate of 3Mbits/sec (384Kbytes/sec) [1]. Suppose that the server uses disks of 100Gbytes of capacity with transference rate of 60Mbytes/sec. In this case, each disk have storage capacity $C = 50$ and load capacity $B = 160$.

We call *single-disk* server, the systems that are constructed in such a way that a entire copy of a movie is done in one disk. But most video servers uses *striped-disks* [1]. In this case, a video is broken into several pieces and each one of these pieces is stored in a different disk. This is done to increase the number of requests that can be attended by the system and to balance the load capacity of the disks. Suppose for example that each disk have transference rate of

60Mbytes/sec and storage capacity of 100Gbytes. Theoretically a disk can support 160 users simultaneously. If we strip the movie along 3 disks, and assume that users requests over the time are distributed uniformly among the three parts of the movie, then the striped-disk can support 480 simultaneously users requests to this movie. For our purposes, we can view each striped-disk as one disk with storage capacity equal to 300Gbytes and load capacity equal to 480. In practice it is better to use striped-disks to balance requests. Consider for example, a single-disk server where a copy of a movie A is in disk 1 and a copy of a movie B is in another disk 2, and there are 320 requests for the movie A and none to the movie B. The system becomes unable to attend 160 requests to the movie A. In a striped-disk system, where the first half part of movie A is stored in disk 1 while the last half part is stored in disk 2, it can attend more users if their requests are distributed along the movie in such a way that requests are divided through the two disks.

We generate classes of instances represented by a tuple (Q, N, T). The value Q corresponds to the number of different movies (different classes) and we consider that $Q \in \{250, 500, 1000\}$. The value N is the number of requests to the movies (number of items), and we assume that $N \in \{5000, 10000, 20000\}$. Finally the value T corresponds to the system type, where T is equal to SC for single-disk system or ST for striped-disk system. In the single-disk system, we have $C = 50$ and $B = 160$, and in the striped-disk system, we have $C = 150$ and $B = 480$.

The requests for movies are generated using the Zipf distribution [14]. This distribution was used previously to generate data for video-on-demand systems [1]. This distribution have the property that the generated data have locality properties. In movie servers it is expected that recent movies are the most requested ones. It is expected that most of the requests goes to a small subset of movies in the server. The Zipf distribution have this property. Let δ be a small positive number. The probability that the n-th movie among Q movies will be requested is p_n given as

$$p_n = \frac{c}{n^{(1+\delta)}}$$

where

$$c = \frac{1}{\sum_{i=1}^{Q}(1/i^{(1+\delta)})}.$$

As δ increases, the distribution becomes more localized and as δ decreases the distribution becomes more uniformly. Considering $Q = 1000$, if $\delta = 0.0$, then 80% of the requests are to approximately 20% of the movies. If $\delta = 1.0$, then 80% of the requests are to approximately 0.3% of the movies. When $\delta = -1.0$ we get the uniform distribution where each movie have the same probability $1/Q$ to be requested.

We present some experimental results in Tables 6.1 and 6.2. All results were obtained in a few seconds. In the tests of these tables, we generate data using $\delta \in \{0.0, 0.5, 1.0\}$, $N \in \{5000, 20000\}$ and $Q \in \{250, 500, 1000\}$. The lower bound is given by $\max\{\lceil Q/C \rceil, \lceil N/B \rceil\}$. In Table 6.1 we consider single-disk system, and in Table 6.2 we consider the striped-disk sys-

tem. We also performed tests with $N = 10000$ but we do not present the results here since we get similar results to the tests with $N = 5000$ and $N = 10000$. We observe that the FFD algorithm generate good results and it becomes better for the striped-disk system. But in comparison with the MW and MW' algorithms it performs worse, since these algorithms generated optimal solutions to all tests. The MW and MW' shows to be very effective algorithms to be used in practical instances to construct video-on-demand servers.

Single-Disk 250 Movies 5000 Requests				500 Movies 5000 Requests			1000 Movies 5000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	FFD	32	32	FFD	34	32	FFD	42	33
	MW	32		MW	32		MW	33	
	MW′	32		MW′	32		MW′	33	
$\delta = 0.5$	FFD	34	32	FFD	39	33	FFD	48	36
	MW	32		MW	33		MW	36	
	MW′	32		MW′	33		MW′	36	
$\delta = 1.0$	FFD	35.8	33	FFD	40.6	34	FFD	50	37.2
	MW	33		MW	34		MW	37.2	
	MW′	33		MW′	34		MW′	37.2	
Single-Disk 250 Movies 20000 Requests				500 Movies 20000 Requests			1000 Movies 20000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	FFD	125	125	FFD	125	125	FFD	127	126
	MW	125		MW	125		MW	126	
	MW′	125		MW′	125		MW′	126	
$\delta = 0.5$	FFD	126	126	FFD	129.4	126	FFD	138	128
	MW	126		MW	126		MW	128	
	MW′	126		MW′	126		MW′	128	
$\delta = 1.0$	FFD	128	126	FFD	133	128	FFD	143	131
	MW	126		MW	128		MW	131	
	MW′	126		MW′	128		MW′	131	

Table 6.1: Performance of the algorithms for Single-Disk.

Striped-Disk 250 Movies 5000 Requests				500 Movies 5000 Requests			1000 Movies 5000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	FFD	11	11	FFD	12	11	FFD	14	11
	MW	11		MW	11		MW	11	
	MW′	11		MW′	11		MW′	11	
$\delta = 0.5$	FFD	12	11	FFD	13	11	FFD	16	12
	MW	11		MW	11		MW	12	
	MW′	11		MW′	11		MW′	12	
$\delta = 1.0$	FFD	12	11	FFD	14	12	FFD	17	13
	MW	11		MW	12		MW	13	
	MW′	11		MW′	12		MW′	13	
Striped-Disk 250 Movies 20000 Requests				500 Movies 20000 Requests			1000 Movies 20000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	FFD	42	42	FFD	42	42	FFD	43	42
	MW	42		MW	42		MW	42	
	MW′	42		MW′	42		MW′	42	
$\delta = 0.5$	FFD	42	42	FFD	43	42	FFD	46	43
	MW	42		MW	42		MW	43	
	MW′	42		MW′	42		MW′	43	
$\delta = 1.0$	FFD	43	42	FFD	45	43	FFD	48	44
	MW	42		MW	43		MW	44	
	MW′	42		MW′	43		MW′	44	

Table 6.2: Performance of the algorithms for Striped-Disk.

In Figures 6.5 to 6.9 we present graphics of the results of the algorithms varying the disk storage capacity. The results are given in the y-axis and the storage capacity of the bin is given in the x-axis. In all these tests we consider the load capacity $B = 160$, the number of different movies $Q = 250$ and the number of requests equal to 5000. In Figure 6.5 (resp. 6.6, 6.7, 6.8, and 6.9) we use δ equal to 1.0 (resp. 0.5, 0.0, -0.5 and -1). In the graphics the MW' algorithm is denoted by $\mathrm{MW2}$. The lower bound is given by $\max\{\lceil Q/C \rceil, \lceil N/B \rceil\}$. Notice that the problem becomes easier as the distribution of requests becomes uniformly, i.e, the value of δ decreases. When $\delta = -1.0$ all algorithms generates solutions almost equal to the lower bound. Another point is that the problem is harder when the capacity is small, as one could expect. When the capacity becomes equal to approximately 10 the algorithms MW and MW' produces optimal solutions. When we consider the capacity greater than 100, the algorithm FFD generates optimal solutions (for δ equal to 1 and 0.5). The MW' algorithm generates better solutions than the MW algorithm in several instances for δ equal to 1.0, 0.5, 0.0 and -0.5. Generally the solutions generated by the algorithm MW' uses 2 or 1 less disks than MW. Most of these better solutions were obtained with capacities between 2 and 8. It is also interesting to notice that the MW algorithm generates a better solution than the MW' algorithm in one test, the one with $\delta = -1$ and capacity equal to 8. In this case the solution found by the MW' algorithm uses 34 disks while the solution generated by the MW algorithm uses 33 disks.

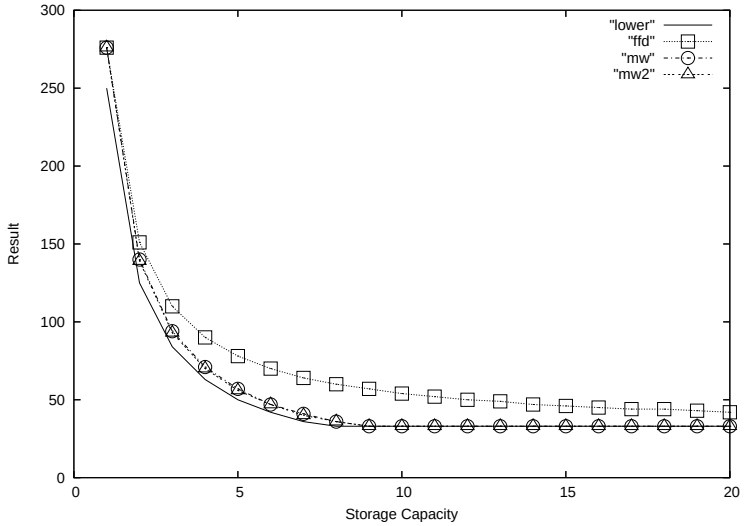

Figure 6.5: Results with $\delta = 1$.

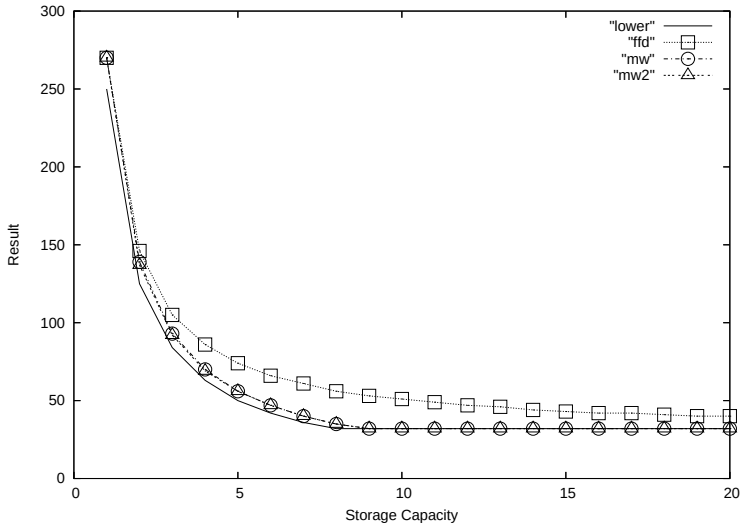

Figure 6.6: Results with $\delta = 0.5$.

6.7 Conclusions and Future Work

In this paper we present approximation algorithms for the online and offline class-constrained bin packing problem. The problem is motivated by applications in the data-placement problem to video-on-demand servers and applications in the cutting and packing area. For the online problem we provide lower bounds for any bounded space algorithm and we also present an algorithm for the unbounded version with approximation factor 2.75. For the offline problem we present practical approximation algorithms for two special cases of the problem, with conditions

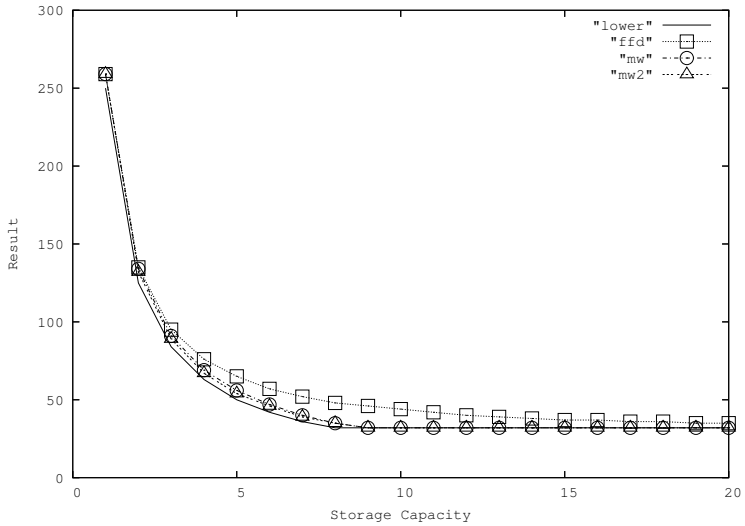

Figure 6.7: Results with $\delta = 0$.

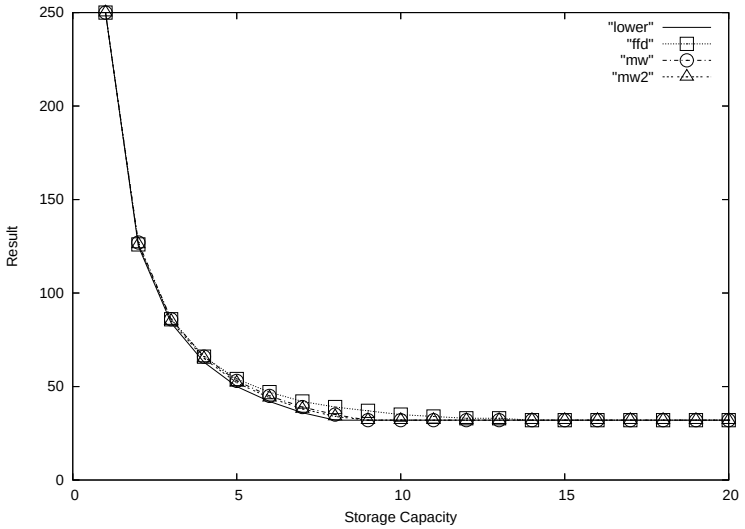

Figure 6.8: Results with $\delta = -0.5$.

already considered in the literature: when all items have the same size and the parameterized version of the problem. We also perform several tests with these practical algorithms. For the instances we considered representing practical ones, the algorithms MW and MW' obtained optimal solutions. At last, we present an APTAS for the special case where the number of different classes of the input instance is bounded by a constant.

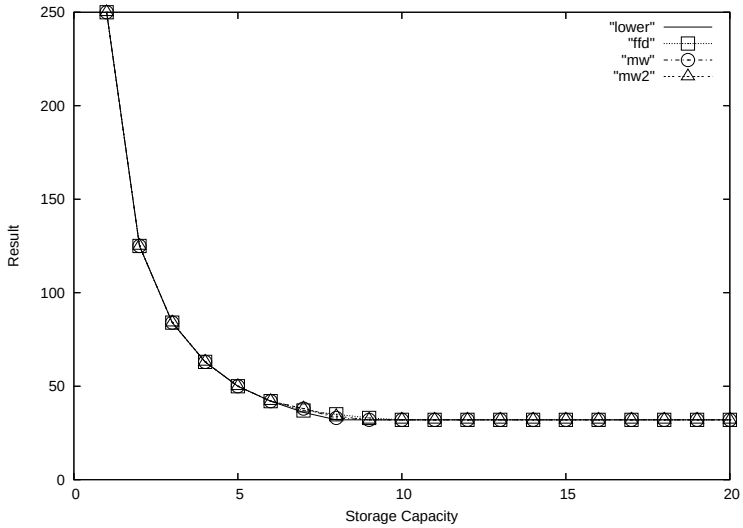

Figure 6.9: Results with $\delta = -1$.

6.8 Bibliography

1. A. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. PhD thesis, University of California at Berkeley, Computer Science Division, 1994.

2. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 2, pages 46–93. PWS, 1997.

3. J. Csirik. The parametric behavior of the first-fit decreasing bin packing algorithm. *Journal of Algorithms*, 15:1–28, 1993.

4. M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. Technical report, IBM, T.J. Watson Research Center, NY, 1998.

5. M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. In *Proceedings of the 1th Brazilian Symposium on Graph Algorithms and Combinatorics*, volume 7 of *Electronic Notes in Dicrete Mathematics*, 2001.

6. W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

7. S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continous media servers. *Parallel Computing Journal*, 24(1):91–122, 1998.

8. L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proceedings of SODA*, pages 223–232, 2000.

9. P. S. Wu J. L. Wolf and H. Shachnai. Disk load balancing for video-on-demand-systems. *ACM Multimedia Systems Journal*, 5:358–370, 1997.

10. D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

11. D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974.

12. J. R. Kalagnanam, M. W. Dawande, M. Trumbo, and H. S. Lee. The surplus inventory matching problem in the process industry. *Operations Research*, 48(4):505–516, 2000.

13. S. R. Kashyap and S. Khuller. Algorithms for non-uniform size data placement on parallel disks. In *Proceedings of FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 265–276, 2003.

14. D. E. Knuth. *The Art of Computer Programming. Volume 3*. Addison-Wesley, 1973.

15. C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. Association Comput. Mach.*, 32(3):562–572, July 1985.

16. F. K. Miyazawa and Y. Wakabayashi. Parametric on-line algorithms for packing rectangles and boxes. *European Journal on Operational Research*, 150:281–292, 2003.

17. M. Peeters and Z. Degraeve. The co-printing problem: A packing problem with a color constraint. *Operations Research*, 52(4):623–638, 2004.

18. S. S. Seiden. On the online bin packing problem. *Journal of ACM*, 49(5):640–671, 2002.

19. H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29:442–467, 2001.

20. H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, 4(6):313–338, 2001.

21. H. Shachnai and T. Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, 32(4):651–678, 2002.

22. H. Shachnai and T. Tamir. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In *Proceedings of 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, RANDOM-APPROX*, volume 2764 of *Lecture Notes in Computer Science*, pages 165–177, 2003.

23. H. Shachnai and T. Tamir. Tight bounds for online class-constrained packing. *Theoretical Computer Science*, 321(1):103–123, 2004.

24. J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, 1971.

25. A. van Vliet. An improved lower bound for online bin packing algorithms. *Inform. Process. Lett.*, 43:277–284, 1992.

26. E. C. Xavier and F. K. Miyazawa. A one-dimensional bin packing problem with shelf divisions. In *2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics*, volume 19 of *Electronic Notes in Discrete Mathematics*, 2005.

27. E. C. Xavier and F. K. Miyazawa. Approximation schemes for knapsack problems with shelf divisions. *Theoretical Computer Sciense*, 352(1-3):71–84, 2006.

28. A. C. Yao. New algorithms for bin packing. *J. Association Comput. Mach.*, 27:207–227, 1980.

Capítulo 7

Artigo: *A Note on the Approximability of Cutting Stock Problems*

G. Cintra[2] F. K. Miyazawa[3] Y. Wakabayashi[4] E. C. Xavier[5]

Abstract

Cutting stock problems and *bin packing* problems are basically the same problems. They differ essentially on the variability of the input items. In the first, we have a set of items, each item with a given multiplicity; in the second, we have simply a list of items (each of which we may assume to have multiplicity 1). Many approximation algorithms have been designed for packing problems; a natural question is whether some of these algorithms can be extended to cutting stock problems. We define the notion of "well-behaved" algorithms and show that well-behaved approximation algorithms for one, two and higher dimensional bin packing problems can be translated to approximation algorithms for cutting stock problems with the same approximation ratios. The results we show include the existence of an asymptotic approximation scheme for the one-dimensional cutting stock problem and an algorithm with an asymptotic performance bound of 2.077 for the two-dimensional cutting stock problem.

Key words: bin packing, cutting stock, approximation algorithm.

[1]This research was partially supported by CNPq (Proc. 478818/03–3, 306526/04–2, 308138/04-0 and 490333/04-4), ProNEx–FAPESP/CNPq (Proc. 2003/09925-5) and CAPES.

[2]Faculdade de Computação e Informática, Universidade Presbiteriana Mackenzie, São Paulo-SP, Brazil.

[3]Corresponding author: Instituto de Computação, Universidade Estadual de Campinas, Caixa Postal 6176, CEP 13084-971 , Campinas-SP, Brazil. fkm@ic.unicamp.br

[4]Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo-SP, Brazil.

[5]Instituto de Computação, Universidade Estadual de Campinas, Campinas-SP, Brazil.

7.1 Introduction

Cutting stock problems are of great interest, both from a theoretical and a practical point-of-view. Their applications go from packing of items into boxes or containers, to cutting of fabrics, hardboards, glasses, foams, etc. The exact computational complexity status of these problems is unknown. It seems that the decision versions of these problems may not be included in NP and that we can only assume that they lie somewhere below EXPSPACE.

In this paper we show that some approximation algorithms for bin packing problems give rise to approximation algorithms for cutting stock problems. More precisely, according to the typology proposed by Wäscher, Haussner and Schumann [18], the problems we consider here are the *Single Bin-Size Bin Packing* (which we abbreviate by SBSBP) and the *Single Stock-Size Cutting Stock* (which we abbreviate by SSSCS). In d*-dimensional* SBSBP problems, we are given a list L of n items, where each item $i \in L$ is a d-dimensional parallelepiped, and we are asked to pack the elements of L into a minimum number of unit-capacity d-dimensional parallelepipeds. The items have to be packed orthogonally and oriented in all dimensions. Furthermore, no two items can overlap in the packing. In d*-dimensional* SSSCS problems, we are given additionally a (positive integer) demand d_i (multiplicity) for each item $i \in L$. Therefore, SBSBP problems can be considered particular cases of SSSCS problems, where all demands are equal to 1. Note, however, that although an instance I for a SSSCS problem can be trivially translated to an instance I' for the corresponding SBSBP problem, the size of I' may be exponential in the size of I. This means that such a trivial translation is not a good approach to tackle SSSCS problems.

We denote by $1\mathrm{SSSCS}$, $2\mathrm{SSSCS}$ and $3\mathrm{SSSCS}$ the one, two and three-dimensional SSSCS problems, respectively; and by $1\mathrm{SBSBP}$, $2\mathrm{SBSBP}$ and $3\mathrm{SBSBP}$ the corresponding SBSBP problems. For the latter, several approximation algorithms have appeared in the literature [12, 7, 3, 1, 13, 6, 4, 5]. Curiously, despite the similarity of the problems, we did not find references to approximation algorithms for SSSCS problems. Pioneering works on these problems were carried out by Gilmore and Gomory [9, 10, 11] in the early sixties, and since them many contributions have appeared [14, 15, 16, 17]. We refer the reader to Cheng et al. [2] for a survey.

In this note we discuss how to extend some approximation algorithms for SBSBP problems to approximation algorithms for SSSCS problems through the notion of "well-behaved" algorithm. In Section 7.3 we consider the $1\mathrm{SSSCS}$ problem. We define the concept of well-behaved algorithm and show that any well-behaved algorithm for the $1\mathrm{SBSBP}$ problem can be translated to an algorithm for the $1\mathrm{SSSCS}$ problem. In Section 7.4 we mention how to obtain similar results for higher dimensional SSSCS problems. We assume that the reader is familiar with the algorithms for the $1\mathrm{SBSBP}$ problem we mention here: NF (Next Fit), FF (First Fit), BF (Best Fit), NFD (Next Fit Decreasing), FFD (First Fit Decreasing), BFD (Best Fit Decreasing), and $\mathrm{H_M}$ (Harmonic).

7.2 Notation

An *instance* $I = (L, s, d)$ of the 1SSSCS problem consists of a list L of elements, in which each element $e \in L$ has size $s_e \in (0, 1]$ and demand $d_e \in \mathbb{Z}^+$; thus $s = (s_e)_{e \in L}$ and $d = (d_e)_{e \in L}$. The *number of items* in the instance I, which we denote by $\|I\|$, is the sum $\sum_{e \in L} d_e$. That is, the number of items is at least the number of elements of L. The *demand* d_e of an element e indicates that there is a multiplicity of d_e *items* of the element of size s_e. We say that an item i corresponding to an element $e \in L$ is an item of *type* e. That is, in the instance I there are d_e items of type e.

For any structure T, we denote by $\langle T \rangle$ the size in bits of the representation of T. Given k lists $Q_1, \ldots, Q_k$, where $Q_i = (a^i_1, \ldots, a^i_{n_i})$, we denote by $Q = Q_1 \| \ldots \| Q_k$ the *concatenation* of these lists, defined as the list $Q = (a^1_1, \ldots, a^1_{n_1}, \ldots, a^k_1, \ldots, a^k_{n_k})$. The number of elements of a list or a set S is denoted by $|S|$.

If $L = (a_1, \ldots, a_n)$ then $expand(L, s, d)$ denotes the list $L' = (s^1_1, \ldots, s^1_{d_1}, \ldots, s^n_1, \ldots, s^n_{d_n})$, where $s^i_j = s(a_i)$, for $1 \leq j \leq d_i$. Given an instance L' of the 1SBSBP problem, we denote by $condense(L')$ the triple (L, s, d), where $L' = expand(L, s, d)$ and $|L|$ is minimum.

For a given instance $I = (L, s, d)$, a one-dimensional bin B can be represented (or described) by a pair (L_B, d_B), where $L_B \subseteq L$, $0 \leq d_B(e) \leq d(e)$ for each $e \in L_B$. We say that such a pair (L_B, d_B) is a *bin type* for I. Clearly, $\langle B \rangle$ is bounded by a polynomial in $\langle I \rangle$.

The *eq-partition* (equal partition) of a list Q is the list $(Q_1, \ldots, Q_k)$, where k is minimum and (i) $Q = (Q_1 \| \ldots \| Q_k)$; (ii) $e' = e''$ for $e', e'' \in Q_i$, $1 \leq i \leq k$. This definition also applies to lists whose items are bins.

7.3 One-dimensional Single Stock-Size Cutting Stock Problem

The one-dimensional single stock-size cutting stock (1SSSCS) problem can be defined as follows:

Problem 1 (1SSSCS) *Given an instance $I = (L, s, d)$ as defined above, find a packing of the items in I into the minimum number of unit-capacity bins.*

A natural approach to obtain approximation algorithms for the 1SSSCS problem is to adapt known algorithms for the 1SBSBP problem. As we mentioned before, the naive approach that transforms a given instance I for the 1SSSCS problem into the list $expand(I)$ and applies an algorithm for the 1SBSBP problem on this list is flawed as both $expand(I)$ and the size of the packing that is produced may be exponential in the size of I. Of course, expansions of I may be easily avoided, so the main concern is whether we can adapt the algorithms so as to

produce solutions with *short descriptions* (that is, descriptions that are polynomial in the size of I). Putting in a more general setting, we would like to address the following question: which properties should an algorithm for the 1SBSBP problem satisfy in order to be transformable into an algorithm for the 1SSSCS problem that produces a packing with a short description? In what follows, we define the notion of well-behaved algorithm, and give an answer to this question.

Definition 7.3.1 *An algorithm $\mathcal{A}'$ that receives an input list L' for the* 1SBSBP *problem is well-behaved if it satisfies the following two properties:*

P1. STABLE ORDER PROPERTY. *The algorithm packs consecutively the equal-sized items that are consecutive in the input list L'. More precisely, if $(L'_1, \ldots, L'_p)$ is an eq-partition of L' then the algorithm packs the items of each L'_i consecutively. Formally, we may consider that the algorithm behaves as follows:*

1.1. *Take $(L'', s, d) := condense(L')$.*

1.2. *Take $L := expand(L''', s, d)$, where L''' is a permutation of L''.*

1.3. *Pack the items following the order given by L.*

P2. GROUPING PROPERTY. *To pack an item, the algorithm does the following.*

2.0 *Suppose $(L_1, \ldots, L_p)$ is an eq-partition of L, where L is the list mentioned in the previous property. The algorithm $\mathcal{A}'$ packs first the list L_1.*

2.1 *Before packing the first item of a list L_i,*

2.1.1. *let $\mathcal{B} = (B_1, B_2, \ldots, B_k)$ be the list of existing non-empty bins, in the order they were generated.*

2.1.2. *Let $(\mathcal{B}_1, \ldots, \mathcal{B}_q)$ be the eq-partition of $\mathcal{B}$.*
Each list $\mathcal{B}_j = (B_j^1, \ldots, B_j^{n_j})$ is said to be a group.

2.1.3. *Let $\mathcal{B}_{q+1}$ be a group with sufficiently many empty bins.*
// New bins are obtained from this group.

2.2. *To pack the first item $e \in L_i$,*

2.2.1. *the algorithm packs e into a bin $B_j^t \in \mathcal{B}_j$, for some j, such that either $j \leq q$ or ($j = q + 1$ and $t = 1$).*

2.2.2. *Now B_j^t becomes the current bin and $\mathcal{B}_j$ the current group.*

2.3 *While the list L_i is non-empty, to pack the next item $e \in L_i$,*

2.3.1. *if possible, packs e into the current bin B_j^t.*

2.3.2. *If $\mathcal{A}'$ fails in the previous step and $B_j^{t+1} \in \mathcal{B}_j$ then $\mathcal{A}'$ packs e into B_j^{t+1}. Now, B_j^{t+1} becomes the current bin.*

2.3.3. *If $\mathcal{A}'$ fails in the previous step, $\mathcal{A}'$ packs e into a bin $B^1_{j'}$, for some group $\mathcal{B}_{j'}$. Now, $B^1_{j'}$ becomes the current bin and $\mathcal{B}_{j'}$ the current group.*

It is not hard to check that NF, FF, BF, $\mathrm{H_M}$, NFD, FFD, and BFD are well-behaved algorithms. Now using this fact, and the concept of a short description of a packing, defined below, we can derive our first result.

Definition 7.3.2 *Let $I = (L, s, d)$ be an instance for the* 1SSSCS *problem and $\mathcal{P}$ a packing of I. A* description *of $\mathcal{P}$ is a list $\mathcal{D}$ of pairs (B, b_B), where $B = (L_B, d_B)$ is a bin type for I and b_B is the multiplicity of the bin type B in the packing $\mathcal{P}$; and if B_e is the number of items of type e in the bin B, then $\sum_{(B,b_B)\in\mathcal{L}} b_B B_e = d_e$ for any $e \in L$. We say that $\mathcal{D}$ is a* short description *if the bin types B are all distinct and $\langle\mathcal{D}\rangle$ is polynomially bounded in $\langle I\rangle$.*

Theorem 7.3.1 *Let I be an instance for the* 1SSSCS *problem and $\mathcal{A}'$ an algorithm for the* 1SBSBP *problem. If $\mathcal{A}'$ is well-behaved, then there exists a polynomial time algorithm $\mathcal{A}$ that produces a packing that is precisely the packing produced by $\mathcal{A}'$ on the list $expand(I)$, differing possibly only on the description of the packing.*

Proof. Let $I = (L, s, d)$, and L' be the permutation of $expand(I)$ that is obtained as a consequence of the stable order property P1, after applying $\mathcal{A}'$ to $expand(I)$. Assume that $(L_1, \ldots, L_k)$ is the eq-partition of L'.

Let $(\mathcal{B}_1, \ldots, \mathcal{B}_q)$ be an eq-partition of the bins generated by algorithm $\mathcal{A}'$ for the items $L_1 \| \ldots \| L_i$ and let $\mathcal{B}_{q+1}$ be a list of sufficiently many empty bins. Clearly, the algorithm $\mathcal{A}$ may use a short description of $(\mathcal{B}_1, \ldots, \mathcal{B}_q)$. Now, consider the packing of the items of the list L_{i+1}. To pack the first item of L_{i+1}, the algorithm chooses a bin B^t_j of a group $\mathcal{B}_j = (B^1_j, \ldots, B^{n_j}_j)$, where $j \leq q + 1$, and tries to pack the items of L_{i+1} in the bins $(B^t_j, \ldots, B^{n_j}_j)$, consecutively. If it fails to pack all items of L_{i+1} in these bins, it continues in the same fashion moving to the first bin of another group.

Suppose that $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_m}$ is the sequence of groups in the list $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_{q+1})$ (of bins) in which the algorithm $\mathcal{A}'$ has packed the items of L_{i+1}, in the order the packing has occurred. Since $\mathcal{A}'$ is a well-behaved algorithm, it packs the items in consecutive bins of each group. First suppose that $m > 1$. In this case, after packing the items of L_{i+1} in the group $\mathcal{B}_{i_1}$, the number of different bins increases by at most 1 (note that the packing of the items may start in any of the bins of the group). After packing items of L_{i+1} in the groups $\mathcal{B}_{i_2}, \ldots, \mathcal{B}_{i_{m-1}}$, the number of different bins does not increase. After packing the remaining items of L_{i+1} in the group $\mathcal{B}_{i_m}$, the number of bins increases by at most 2. Therefore, the number of different bins after packing the whole list L_{i+1} increases by at most 3. When $m = 1$, the number of different bins increases by at most 2.

Notice that with a simple calculation, the algorithm $\mathcal{A}$ can figure out how many items of L_{i+1} can be packed in a bin of a group $\mathcal{B}_j$ and how many bins of this group it uses to pack

these items. After packing all the lists $L_1, \ldots, L_k$, we can conclude that the number of different bins is at most $3k$. This shows that (mimicking the behavior of algorithm $\mathcal{A}'$) we may design a polynomial time algorithm $\mathcal{A}$ that produces a packing that has a short description. □

Denote by $\mathrm{NF}_{\mathrm{cs}}$, $\mathrm{FF}_{\mathrm{cs}}$, $\mathrm{BF}_{\mathrm{cs}}$, $\mathrm{H}_{\mathrm{Mcs}}$, $\mathrm{NFD}_{\mathrm{cs}}$, $\mathrm{FFD}_{\mathrm{cs}}$ and $\mathrm{BFD}_{\mathrm{cs}}$ the algorithms NF, FF, BF, H_{M}, NFD, FFD and BFD, respectively, adapted for the 1SSSCS problem that generate packings with short descriptions.

Corollary 7.3.2 *The algorithm* $\mathrm{NF}_{\mathrm{cs}}$ *(respectively* $\mathrm{NF}_{\mathrm{cs}}$, $\mathrm{FF}_{\mathrm{cs}}$, $\mathrm{BF}_{\mathrm{cs}}$, $\mathrm{H}_{\mathrm{Mcs}}$, $\mathrm{NFD}_{\mathrm{cs}}$, $\mathrm{FFD}_{\mathrm{cs}}$ *and* $\mathrm{BFD}_{\mathrm{cs}}$*) has asymptotic performance bound* 2 *(respectively* 1.7, 1.7, $1.691\ldots$, $1.691\ldots$, $11/9$ *and* $11/9$*). The bound for* $\mathrm{H}_{\mathrm{Mcs}}$ *holds when* $M \to \infty$.

Considering the same ideas of short descriptions presented for the well-behaved algorithms, we may also convert the AFPTAS of Fernandez de la Vega and Lueker [7] into an AFPTAS for the 1SSSCS problem. That is, the following result holds.

Theorem 7.3.3 *There exists an* AFPTAS *for the* 1SSSCS *problem.*

7.4 Two and Higher Dimensional Single Stock-Size Cutting Stock Problems

An *instance* $I = (L, w, h, d)$ for the 2SSSCS problem consists of a list of elements L, each element $e \in L$ with width $w_e \in (0, 1]$, height $h_e \in (0, 1]$ and demand $d_e \in \mathbb{Z}^+$. Most of the notation we used in the context of the 1SSSCS problem can be extended easily to the context of 2SSSCS, as for example, $expand(L, w, h, d)$, $condense(L)$, etc. For this problem we can also define the concept of well-behaved algorithm. Although better definitions may be given, we present a simple definition of *well-behaved algorithm* for the 2SSSCS problem, as this can be extended easily to higher dimensions.

Definition 7.4.1 *An algorithm* $\mathcal{A}$ *that receives an input list* L' *for the problem* 2SBSBP *is well-behaved if it satisfies the following properties:*

Q1. STABLE ORDER PROPERTY. *The behavior of the algorithm can be described as follows:*

- **1.1.** *Take* $(L'', w, h, d) := condense(L')$.
- **1.2.** *Take* $L := expand(L''', s, d)$*, where* L''' *is a permutation of* L''.
- **1.3.** *Pack the items following the order given by* L.

Q2. LEVEL ORIENTED PROPERTY. *The strategy used by the algorithm to produce a packing is the following:*

2.1 *The algorithm generates a list $\mathcal{L}$ of levels using a well-behaved algorithm for the* 1SBSBP *problem.*

2.2 *The algorithm uses a well-behaved algorithm for the* 1SBSBP *problem to pack the levels of $\mathcal{L}$ into unit-capacity two-dimensional bins.*

Theorem 7.4.1 *Let I be an instance for the* 2SSSCS *problem and $\mathcal{A}'$ an algorithm for the* 2SBSBP *problem. If $\mathcal{A}'$ is a well-behaved algorithm, then there exists a polynomial time algorithm $\mathcal{A}$ that produces a packing that is precisely the packing produced by the algorithm $\mathcal{A}'$ on the list $expand(I)$, differing possibly only on the description of the packing.*

One of the most famous algorithm for the 2SBSBP problem is the algorithm HFF (Hybrid First Fit), presented by Chung, Garey and Johnson [3]. These authors proved that HFF has an asymptotic performance bound of 2.125, and later Caprara [1] proved that this algorithm has an asymptotic performance bound of $2.077\ldots$. Frenk and Galambos [8] proved that the next fit variant of the algorithm HFF, which we denote by HNF, has an asymptotic performance bound of $3.382\ldots$. The algorithm with the best known asymptotic performance bound for 2SBSBP, which we denote by HC, is due to Caprara [1] and has bound $1.691\ldots$. These three algorithms are hybrid and use algorithms for the 1SBSBP problem to pack items into levels and levels into two-dimensional bins. Moreover, all algorithms for the 1SBSBP problem used as subroutines have a corresponding version for the 1SSSCS problem, given by Corollary 7.3.2 or by Theorem 7.3.3.

Corollary 7.4.2 *There exists an algorithm* HNF_{cs} *(resp.* HFF_{cs}, HC_{cs}*) with asymptotic performance bound* $3.382\ldots$ *(resp.* $2.077\ldots$, $1.691\ldots$*) for the* 2SSSCS *problem.*

Most of the ideas presented here can also be extended to higher dimensions. In particular, the 4.84-approximation algorithms of Li and Cheng [13] and of Csirik and van Vliet [6] can be translated to algorithms for the 3SSSCS problem, as they generate packings that consist of levels. We can prove that these algorithms are well-behaved and that the following holds.

Corollary 7.4.3 *There exist algorithms for the problem* 3SSSCS *with asymptotic performance bound* 4.84.

7.5 Bibliography

1. A. Caprara, Packing 2-dimensional bins in harmony, In Proceedings of 43rd Symposium on Foundations of Computer Science, IEEE Computer Society Press, 2002, pp. 490–499.

2. C. H. Cheng, B. R. Feiring, and T. C. E. Cheng, The cutting stock problem – a survey, International Journal of Production Economics, 36(3)(1994) 291–305.

3. F. R. K. Chung, M. R. Garey, and D. S. Johnson, On packing two-dimensional bins, SIAM Journal on Algebraic and Discrete Methods, 3(1982) 66–76.

4. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, Approximation algorithms for bin packing - an updated survey, in: G. Ausiello, M. Lucertini, and P. Serafini (Eds.), Algorithms design for computer system design, Springer-Verlag, New York, 1984, pp. 49–106.

5. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, Approximation algorithms for bin packing: a survey, in: D. Hochbaum, (Eds.), Approximation Algorithms for NP-hard Problems, PWS Publishing Company, Boston, 1997, pp. 46–93.

6. J. Csirik and A. van Vliet, An on-line algorithm for multidimensional bin packing, Operations Research Letters, 13(3)(1993) 149–158.

7. W. Fernandez de la Vega and G. S. Lueker, Bin packing can be solved within $1 + \epsilon$ in linear time, Combinatorica, 1(4)(1981) 349–355.

8. J. B. Frenk and G. Galambos, Hybrid next fit algorithm for the two-dimensional rectangle bin packing problem, Computing, 39(1987) 201–217.

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

10. P. Gilmore and R. Gomory, A linear programming approach to the cutting stock problem - part II, Operations Research, 11(1963) 863–888.

11. P. Gilmore and R. Gomory, Multistage cutting stock problems of two and more dimensions, Operations Research, 13(1965) 94–120.

12. N. Karmarkar and R. M. Karp, An efficient approximation scheme for the one dimensional bin packing problem, In Proceedings of the 23rd Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1982, pp. 312–320.

13. K. Li and K-H. Cheng, A generalized harmonic algorithm for on-line multidimensional bin packing, Technical Report UH-CS-90-2, University of Houston, 1990.

14. G. Scheithauer and J. Terno, The modified integer round-up property of the one-dimesional cutting stock problem, European Journal of Operational Research, 84(3)(1995) 562–571.

15. P. H. Vance, Branch-and-price algorithms for one-dimensional cutting stock problem, Computational Optimization and Applications, 9(3)(1998) 211–228.

16. F. Vanderbeck, Computational study of a column generation algorithm for bin packing and cutting stock problems, Mathematical Programming, 86(3)(1999) 565–594.

17. G. Wäscher, An LP-based approach to cutting stock problems with multiple objectives, European Journal of Operational Research, 44(2)(1990) 175–184.

18. G. Wäscher, H. Haussner, and H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research, this issue, 2006.

Capítulo 8

Artigo: *Algorithms for Two-Dimensional Cutting Stock and Strip Packing Problems Using Dynamic Programming and Column Generation*

G. Cintra[2] F. K. Miyazawa[3] Y. Wakabayashi[4] E. C. Xavier[5]

Abstract

We investigate several two-dimensional guillotine cutting stock problems. We restrict our attention to the variants of these problems where the cuts are k-staged. We also consider the variants in which orthogonal rotations are allowed. We first present a dynamic programming based algorithm for the *Rectangular Knapsack* (RK). Using this algorithm we solved all instances of the RK problem found at the OR–LIBRARY, including one for which no optimal solution was known. We also consider the *Two-dimensional Cutting Stock* (2CS) problem. We present a column generation based algorithm for this problem that uses the first algorithm above mentioned to generate the columns. We also investigate a variant of this problem where the bins have different sizes. At last, we study the *Two-dimensional Strip Packing* (SP) problem. We

[1]This research was partially supported by CNPq (Proc. 478818/03–3, 306526/04–2, 308138/04-0 and 490333/04-4), ProNEx–FAPESP/CNPq (Proc. 2003/09925-5) and CAPES.

[2]Faculdade de Computação e Informática, Universidade Presbiteriana Mackenzie, São Paulo-SP, Brazil.

[3]Instituto de Computação, Universidade Estadual de Campinas, Campinas-SP, Brazil.

[4]Corresponding author: Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo-SP, Brazil. yw@ime.usp.br

[5]Instituto de Computação, Universidade Estadual de Campinas, Campinas-SP, Brazil.

also present a column generation based algorithm for this problem that uses the second algorithm above mentioned where staged patterns are imposed. In this case we solve instances for two-, three- and four-staged patterns. We report on some computational experiments with the algorithms of this paper. The results indicate that these algorithms seem to be suitable for solving real-world instances. We give a detailed description (a pseudo-code) of all the algorithms presented here, so that the reader may easily implement these algorithms.

Key words: column generation, cutting stock, guillotine cutting, dynamic programming, two-dimensional packing, strip packing

8.1 Introduction

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. Very often, the large objects (bins) and the small objects (items) are two-dimensional 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. Another usual restriction for these problems are the staged cuts. A k*-staged cutting* is a sequence of at most k stages of cuts, each stage of which is a set of parallel guillotine cuts, performed on the objects obtained in the previous stage. Clearly, the cuts in each stage must be orthogonal to the cuts in the previous stage. We assume, without loss of generality, that the cuts are infinitely thin.

In what follows, we define the problems we consider in this paper. In all of them, we assume that at most k stages of guillotine cuts are allowed, even if is is not explicitely mentioned.

In the *Rectangular Knapsack* (RK) problem we are given a rectangle $B = (W, H)$ with width W and height H, and a list of m items (types of rectangles), each item i with width w_i, height h_i, and value v_i ($i = 1, \ldots, m$). We wish to determine how to cut the rectangle B, so as to maximize the sum of the values of the items that are produced. We assume that many copies of the same item can be produced. We denote such an instance by $I = (W, H, w, h, v)$. Here, as well in the next problems, we assume that $w = (w_1, \ldots, w_m)$, $h = (h_1, \ldots, h_m)$, and $d = (d_1, \ldots, d_m)$ are lists. We use $(\ )$ to represent an empty list and the operator $\|$ to concatenate lists.

The *Two-dimensional Cutting Stock* (2CS) problem is defined as follows. Given an unlimited quantity of two-dimensional bins $B = (W, H)$, with width W and height H, and a list of m items (small rectangles) each item i with dimensions (w_i, h_i) and demand d_i ($i = 1, \ldots, m$), determine how to cut the smallest number of bins B so as to produce d_i unities of each item i. An instance for the 2CS problem is denoted by $I = (W, H, w, h, d)$.

We also consider the 2CS problem with variable bin sizes, that will be denoted by BPV.

This problem is similar to the previous one: the difference is that we are now given a list of two-dimensional bin types $B_1, \ldots, B_b$, each bin type B_j with dimensions (W_j, H_j) and value V_j (there is an unlimited quantity of them). We want to determine how to produce d_i unities of each item i, $1 \leq i \leq m$, so as to minimize the sum of the values of the bins that are used. Such an instance for this problem is denoted by $I = (W, H, V, w, h, d)$, where $W = (W_1, \ldots, W_b)$, $H = (H_1, \ldots, H_b)$ and $V = (V_1, \ldots, V_b)$.

The *Two-dimensional Strip Packing* (SP) problem is the following: given a two-dimensional strip with width W and infinite height, and a list of m items (rectangles), each item i with dimensions (w_i, h_i) and demand d_i, $1 \leq i \leq m$, determine how to produce d_i unities of each item i from the strip, so as to minimize the height of the part of the strip that is used. We require that the cuts be k-staged, and that in the first stage (in which horizontal cuts are performed) the distance between any two subsequent cuts must be at most H (a commom restriction in practice, imposed by the cutting machines). An instance as above will be denoted by $I = (W, H, w, h, d)$.

For all these problems, we consider variants with orthogonal rotations. Unless otherwise stated, we assume that the items are oriented (that is, rotations of the items are not allowed). The variants of these problems in which the items may be rotated orthogonally are denoted by RK^r, BP^r, BPV^r and SP^r. We also assume that, in all instances the items have feasible dimensions, that is, each of them fit into the given bin (or some bin type) or strip.

This paper focuses on algorithms for the problems above mentioned. They are classical hard optimization problems, interesting both from theoretical as well as practical point-of-view. Most of them have been largely investigated. In the next sections we discuss these problems and mention some of the results that have appeared in the literature.

We call each possible way of cutting a bin a *cutting pattern* (or simply *pattern*). To represent the patterns (and the cuts to be performed) we adopt the convention that is generally used in this context. We consider the Euclidean plane $\mathbb{R}^2$, with the xy coordinate system, and assume that the width of a rectangle is represented in the x-axis, and the height is represented in the y-axis. We also assume that the position $(0, 0)$ of this coordinate system represents the bottom left corner of the bin. Thus a bin of width W and height H corresponds to the region defined by the rectangle whose bottom left corner is at the position $(0, 0)$ and the top right corner is at the position (W, H). To specify the position of an item i in the bin, we specify the coordinates of its bottom left corner. Using these conventions, it is not difficult to define more formally what is a pattern and how we can represent one.

A *guillotine pattern* is a pattern that can be obtained by a sequence of guillotine cuts applied to the original bin and to the subsequent small rectangles that are obtained after each cut (see Figure 8.1).

Many practical applications have restrictions on the number of cutting stages to obtain the final items, especially when the cost of the material to be cut is low compared to the industrial cost involved in the cutting process. We say that a pattern is k-*staged* if it is obtained after

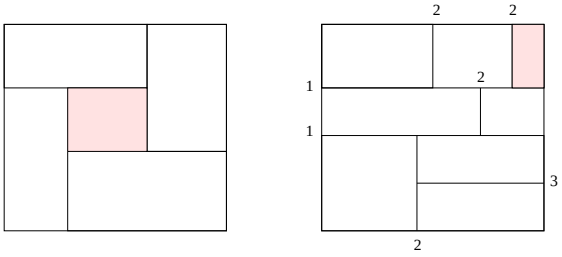

Figure 8.1: (a) Non-guillotine pattern; (b) Guillotine pattern.

performing k stages of cutting (an eventual additional stage is allowed in order to separate an item from a wasted area). In Figure 8.1(b) we have a 3-staged (guillotine) pattern (We consider that the gray area is a wasted area). Following other papers in the literature (see [12, 13, 46]), we assume that the first cutting stage is performed in the horizontal direction, for all problems on staged patterns.

This paper is organized as follows. In Section 8.2, we focus on the Rectangular Knapsack (RK) problem, where we present dynamic programming based algorithms to obtain exact solutions for it.

Section 8.3 is devoted to the Two-dimensional Cutting Stock (2CS) problem. We describe two algorithms for it, both based on the column generation approach. One of them uses a perturbation strategy to deal with the residual instances. We also consider the variant of the 2CS problem in which orthogonal rotations are allowed. In Section 8.4 we study the BPV problem, a variant of the 2CS problem where bins may have different sizes and values. In Section 8.5 we study the Strip Packing (SP) problem. All algorithms based on the column generation approach we present here make use of the exact algorithms of Section 8.2.

Finally, in Section 8.6 we report on the computational results we have obtained with the presented algorithms, and in the last section we make some final remarks. The computational tests show that the algorithms we describe here find solutions for medium size instances that are very close to the optimum in small amount of time.

Observation: The results of this paper is an extension of the work done by Cintra [18], where he presented column generation algorithms for the non-staged versions of the problems RK, 2CS and BPV. In this paper we extend his work to consider staged patterns and also to the SP problem. A paper containing the results presented here and the results presented by Cintra [18] was submitted to a journal.

8.2 The k-staged rectangular knapsack problem

The *Rectangular Knapsack* (RK) problem has been largely investigated since the sixties. Gilmore and Gomory [27, 28] studied this problem (on guillotine cuts) and they also introduced in 1965

the variant with k-staged cuts [29]. In 1972, Herz [31] presented a recursive algorithm to obtain patterns, called canonical, making use of the so-called *discretization points*. Christofides and Whitlock [15] showed a dynamic programming approach to compute the discretization points. Some papers also consider exact tree search procedures [7, 39] for this problem. Arenales and Morábito [3] proposed an exact branch and bound algorithm using an and-or-graph search approach for non-guillotine patterns.

Wang [47] proposed an algorithm that generates cutting patterns combining smaller pieces of patterns. Beasley [5] proposed a dynamic programming approach using the discretization points of Herz for both the non-staged and the staged versions of the problem. Recently, Belov and Scheithauer [8] presented a branch and cut algorithm for a variant restricted to 2-staged (oriented) patterns. Lodi and Monaci [36] also investigated the 2-staged version. For the variant in which all items must be packed at most once, Jansen [33] obtained a $(2+\epsilon)$-approximation algorithm.

We describe now the algorithms we implemented for the RK problem. For that, we present first some concepts and results. We basically implement the recurrence formulas proposed by Beasley (using dynamic programming) combined with the concept of discretization points defined by Herz [31]. This approach seems to be very effective: we could solve an instance of the OR-Library whose optimal solution was unknown.

Let $I = (W, H, w, h, v)$ be an instance of the RK problem. We consider that W, H, and the entries of w and h are all integer numbers. If this is not the case, we can obtain an equivalent integral instance simply by multiplying the widths and/or the heights of the bin and of the items by appropriate numbers.

A *discretization point of the width* (respectively of the *height*) is a value $i \leq W$ (respectively $j \leq H$) that can be obtained by an integer conic combination of $w_1, \ldots, w_m$ (respectively $h_1, \ldots, h_m$).

We denote by P (respectively Q) the set of all discretization points of the width (respectively height). Following Herz, we say that a *canonical pattern* is a pattern for which all cuts are made at discretization points.

We note that it suffices to consider only canonical patterns (for every pattern that is not canonical there is an equivalent one that is canonical). To refer to them, the following functions will be useful. For a rational $x \leq W$, let $p(x) := \max(i \mid i \in P,\ i \leq x)$ and for a rational $y \leq H$, let $q(y) := \max(j \mid j \in Q,\ j \leq y)$.

We denote by $V(W, H, k, \mathcal{V})$, (respectively $V(W, H, k, \mathcal{H})$) the value of an optimal canonical guillotine k-staged pattern for a rectangle of dimensions (W, H) where the first stage of cut is done in the vertical (horizontal) direction, i.e, the parameters $\mathcal{H}$ and $\mathcal{V}$ indicate the direction of the first cutting stage: either horizontal or vertical. The recurrence formulas to calculate these values are given in what follows. In this formula, $v(w, h)$ denotes the value of the most valuable item that can be cut in a rectangle of dimensions (w, h); it is 0 if no item can be cut in such a

rectangle.

$V(w,h,0,\mathcal{V} \text{ or } \mathcal{H}) = v(w,h)$
$V(w,h,k,\mathcal{V}) = \max\{V(w,h,k-1,\mathcal{H}), (V(w',h,k-1,\mathcal{H}) + V(p(w-w'),h,k,\mathcal{V}) \mid w' \in P,\ w' \leq w/2\},$
$V(w,h,k,\mathcal{H}) = \max\{V(w,h,k-1,\mathcal{V}), (V(w,h',k-1,\mathcal{V}) + V(w,q(h-h'),k,\mathcal{H}) \mid h' \in Q,\ h' \leq h/2)\}.$

8.2.1 Discretization points

In this section we present, for completeness, an algorithm, called DDP (Discretization using Dynamic Programming) to find the discretization points of the width (or height). The algorithm is already known in the literature and a detailed description of this algorithm and other ones to generate discretization points can be found in [18].

The presented algorithm finds the discretizations points of the width. To find the discretization points of the height, it is only needed to consider the height of the items, inspite of the width and to consider the height of the bin. The basic idea of this algorithm is to solve a knapsack problem in which every item i has weight and value w_i ($i = 1, \ldots, m$), and the knapsack has capacity W. The well-known dynamic programming technique for the knapsack problem (see [23]) finds optimal values of knapsacks with (integer) capacities taking values from 1 to W. It is easy to see that j is a discretization point if and only if the knapsack with capacity j has optimal value j.

Input: W, w_1, . . ., w_m.
Output: a set $\mathcal{P}$ of discretization points.
$\mathcal{P} = \{0\}$.
For j = 0 to W do $c_j = 0$.
For i = 1 to m do
  For j = w_i to W
    If $c_j < c_{j-w_i} + w_i$ then $c_j = c_{j-w_i} + w_i$
For j = 1 to W
  If $c_j = j$ then $\mathcal{P} = \mathcal{P} \cup \{j\}$.
Return $\mathcal{P}$.

Algorithm 8.1: DDP

We note that the algorithm DDP requires time $O(mW)$. The algorithm DDP is suited for instances in which W is not very large. In all the computational tests, presented in Section 8.6, we used the algorithm DDP to generate the discretization points.

8.2.2 The k-staged RK problem

In this section we present an exact algorithm to solve this problem and a variant where rotations are allowed.

Let $I = (W, H, w, h, v)$, with $w = (w_1, \ldots, w_m)$, $h = (h_1, \ldots, h_m)$ and $v = (v_1, \ldots, v_m)$, be an instance of the problem RK.

We denote by P (respectively Q) the set of all discretization points of the width (respectively height). We denote by $v(w, h)$ the value of the most valuable item that can be cut (or be obtained without any cut) from a rectangle of dimensions (w, h), or 0 if no item can be cut (or be obtained).

We describe in the sequel the algorithm SDP (Algorithm 8.2) that solves the recurrence formulas proposed for the k-staged RK problem. In the description of this algorithm we assume that the first stage of cuts is done in the horizontal direction.

Let w_{min} (respectively h_{min}) be the minimum width (height) of the items in the instance. Let P_0 be the set of values $i \in P$ such that $i \leq W - w_{min}$, and let Q_0 be the set of values $j \in Q$ such that $j \leq H - h_{min}$. Let $P_1 = P_0 \cup \{W\}$, and let $Q_1 = Q_0 \cup \{H\}$. We can use the sets P_1 and Q_1 instead of the sets P and Q in the above recurrence and possibly obtain an improvement in the time to solve it, since no item can be to the right (respectively to the top) of a vertical (respectively horizontal) cut done in a position greater than $W - w_{min}$ ($H - h_{min}$).

We have designed the algorithm in such a way that a pattern corresponding to an optimal solution can be easily obtained. For that, the algorithm stores in a matrix, for every rectangle of width $p_i \in P_1$ and height $q_j \in Q_1$, which is the direction (horizontal or vertical) and the position of the first guillotine cut that has to be made in this rectangle. In case no cut should be made in the rectangle, the algorithm stores the item that corresponds to this rectangle.

When the algorithm SDP halts, we have that $V(k, i, j)$ contains the optimal value that can be obtained in k stages for a rectangle with dimensions (p_i, q_j). Furthermore, $guillotine(k, i, j)$ indicates the direction of the first guillotine cut, and $position(k, i, j)$, stores the corresponding position (in the x-axis or in the y-axis) of the first guillotine cut. If $guillotine(k, i, j) = nil$, then no cut has to be made in this rectangle. In this case, $item(i, j)$ (if nonzero) indicates which item corresponds to this rectangle. The value of the optimal solution will be in $V(k, r, s)$, where $r = |P_1|$ and $s = |Q_1|$.

The algorithm calculates the best solutions for the 1-staged problem and then uses this information to calculate the best solutions of the 2-staged problem and so forth. There may be a stage in which no cut has to be made: that happens when the best solution of a given stage, say l, is the best solution of the previous stage $l - 1$. In this case, the value *'P'* is stored in the corresponding entry of $guillotine$, indicating that the solution is given by the previous stage.

Consider that $r > s$. The attributions of value to the variable t can be done in $O(\log r)$ time by using binary search in the set of the discretization points. But we can use a vector X (resp. Y), of size W (resp. H), and let X_i (resp. Y_j) contain $p(i)$ (resp. $q(j)$). Once the discretization

Input: An instance $I = (W, H, w, h, v, k)$ of the k-staged RK problem.
Output: An optimal k-staged solution for I.

Let $p_1 < \ldots < p_r$, be the points in P_1.
Let $q_1 < \ldots < q_s$, be the points in Q_1.
For i = 1 to r
  For j = 1 to s
    $V(0, i, j) = \max(\{v_f \mid 1 \leq f \leq m,\ w_f \leq p_i \text{ and } h_f \leq q_j\} \cup \{0\})$.
    $item(0, i, j) = \max(\{f \mid 1 \leq f \leq m,\ \ w_f \leq p_i,\ h_f \leq q_j \text{ and } v_f = V(1, i, j)\} \cup \{0\})$.
    $guillotine(0, i, j) = nil$.
If k is even then $A =$ 'H' else $A =$ 'V'
For $l = 1$ to k
  For i = 2 to r
    For j = 2 to s
      $V(l, i, j) = V(l-1, i, j)$
      $guillotine(l, i, j) =$'P'
      If $A =$ 'V' then
        $n = \max(f \mid 1 \leq f \leq s \text{ and } q_f \leq \lfloor \frac{q_j}{2} \rfloor)$.
        For y = 1 to n
          $t = \max(f \mid 1 \leq f \leq s \text{ and } q_f \leq q_j - q_y)$.
          If $V(l, i, j) < V(l-1, i, y) + V(l, i, t)$ then
            $V(l, i, j) = V(l-1, i, y) + V(l, i, t)$, $position(l, i, j) = q_y$ and $guillotine(l, i, j) =$ 'H'.
      Else
        $n = \max(f \mid 1 \leq f \leq r \text{ and } p_f \leq \lfloor \frac{p_i}{2} \rfloor)$.
        For x = 1 to n
          $t = \max(f \mid 1 \leq f \leq r \text{ and } p_f \leq p_i - p_x)$.
          If $V(l, i, j) < V(l-1, x, j) + V(l, t, j)$ then
            $V(l, i, j) = V(l-1, x, j) + V(l, t, j)$, $position(l, i, j) = p_x$ and $guillotine(l, i, j) =$ 'V'.
  If $A =$ 'V' then $A =$ 'H' else $A =$ 'V'.

Algorithm 8.2: SDP

points are calculated, it requires time $O(W + H)$ to determine the values in the vectors X and Y. Using these vectors, each attribution to the variable t can be done in constant time and leads to an implementation of the algorithm DP, using DDP as a subroutine, with time complexity $O(mW + mH + (\frac{k}{2})(\frac{r^2 s}{2}) + (\frac{k}{2})(\frac{rs^2}{2}))$. In any case, the amount of memory required by the algorithm SDP is $O(krs + W + H)$. We use this strategy in our implementation.

We can also use the algorithm SDP to solve the k-staged RK^r problem, in which orthogonal

rotations of the items are allowed. For that, for each item i in I, of width w_i, height h_i and value v_i, we add another item of width h_i, height w_i and value v_i, whenever $w_i \neq h_i$, $w_i \leq H$ and $h_i \leq W$. We denote the corresponding algorithm for this case by SDP^r.

8.3 The 2CS problem

We focus now on the *Two-dimensional Cutting Stock* (2CS) problem. Gilmore and Gomory [27, 28, 29] in the early sixties were the first to propose the use of the column generation approach for this problem. They proposed the k-staged pattern version and also considered the BPV problem, the variant of 2CS with bins of different sizes.

Alvarez-Vales, Parajon and Tamarit [2] also presented a column generation approach for the 2CS problem. They used the dynamic programming algorithm presented by Beasley and also some meta-heuristic procedures. Puchinger and Raidl [42] investigated the 3-staged version: they applied the column generation approach using either a greedy heuristic or an evolutionary algorithm to generate columns.

Riehme, Scheithauer and Terno [44] designed an algorithm for the 2CS problem with *extremely varying order demands*. Their algorithm is also based on the column generation approach and is restricted to a 2-staged problem. Vanderbeck [46] also proposed a column generation approach for a cutting stock problem with several different restrictions. The solution must be 3-staged and unused parts of some stock can be used later as a new stock. The problem involves other practical restrictions.

For the special case in which the demands are all equal to 1 (also known as bin packing problem) Chung, Garey and Johnson [16] presented the first approximation algorithm for this problem, called HFF (Hybrid First Fit), shown to have asymptotic performance bound at most 2.125. Later, Caprara [11] proved that HFF has asymptotic performance bound at most 2.077; and he also presented an 1.691-approximation algorithm (this is the best known result for this problem). We observe that the algorithm HFF is a 2-staged algorithm, and therefore may be used as a subroutine to any k-staged problem for $k \geq 2$. These results are for the oriented case. When orthogonal rotations are allowed, Miyazawa and Wakabayashi [38] presented a 2.64-approximation algorithm. For the particular case in which all bins are squares and rotations are allowed, Epstein [25] presented a 2.45-approximation algorithm. In [19], we have shown that some of the approximation algorithms for the bin packing problem can be modified for the cutting stock problem. In this case the algorithms are of polynomial time and preserve the same approximation factor of the original algorithms.

The column generation algorithms we presented in this section were developed by Cintra [18], but for completeness we also present these algorithms here.

To discuss the column generation approach, let us first formulate the 2CS problem as an ILP (Integer Linear Program). Let $I = (W, H, w, h, d)$ be an instance for the 2CS problem.

Represent each pattern j for the instance I as a vector p_j, whose i-th entry indicates the number of times item i occurs in this pattern. The 2CS problem consists then in deciding how many times each pattern has to be used to meet the demands and minimize the total number of bins that are used.

Let n be the number of all possible patterns for I, and let P denote an $m \times n$ matrix whose columns are the patterns $p_1, \ldots, p_n$. If we denote by d the vector of the demands, then we have the following ILP formulation: minimize $\sum_{j=1}^{n} x_j$ subject to $Px = d$ and $x_j \geq 0$ and x_j integer for $j = 1, \ldots, n$. (The variable x_j indicates how many times the pattern j is selected.)

The well-known column generation method proposed by Gilmore and Gomory [27] consists in solving the relaxation of the above ILP, shown below. The idea is to start with a few columns and then generate new columns of P, only when they are needed.

$$\begin{array}{ll} \text{minimize} & x_1 + \ldots + x_n \\ \text{subject to} & Px = d \\ & x_j \geq 0 \quad j = 1, \ldots, n. \end{array} \tag{8.1}$$

We can use the algorithm SDP (for the RK problem) to generate new columns (with k-staged guillotine patterns). If y_i is the dual value corresponding to each item i, $1 \leq i \leq m$, then we want a pattern that maximizes $\sum_{i=1}^{m} y_i z_i$, where z_i is the number of times item i is used in the pattern. We describe in the sequel the algorithm SimplexCG$_2$ that solves (8.1) (Algorithm 8.3).

Input: An instance $I = (W, H, w, h, d)$ of the 2CS problem.
Output: An optimal solution for (8.1), where the columns of P are patterns for I.
Subroutine: The algorithm SDP for the RK problem.

1 Let $x = d$ and B be the identity matrix of order m.
2 Solve $y^T B = \mathbb{1}^T$.
3 Generate a new column z executing the algorithm SDP with parameters W, H, w, h, y.
4 If $y^T z \leq 1$, return B and x and halt (x corresponds to the columns of B).
5 Otherwise, solve $Bw = z$.
6 Let $t = \min\left(\frac{x_j}{w_j} \mid 1 \leq j \leq m,\ w_j > 0\right)$.
7 Let $s = \min\left(j \mid 1 \leq j \leq m,\ \frac{x_j}{w_j} = t\right)$.
8 For $i = 1$ to m do
  8.1 $B_{i,s} = z_i$.
  8.2 If $i = s$ then $x_i = t$; otherwise, $x_i = x_i - w_i t$.
9 Go to step 2.

Algorithm 8.3: SimplexCG$_2$

We implemented this algorithm using subroutine SDP described in Section 8.2.

We describe now a procedure to find an integer solution from the solutions obtained by SimplexCG$_2$. The procedure is iterative. Each iteration starts with an instance I of the 2CS problem and consists basically in solving (8.1) with SimplexCG$_2$ obtaining B and x. If x is integral, we return B and x and halt. Otherwise, we calculate $x^* = (x_1^*, \ldots, x_m^*)$, where $x_i^* = \lfloor x_i \rfloor$ ($i = 1, \ldots, m$). For this new solution, possibly part of the demand of the items is not fulfilled. More precisely, the demand of each item i that is not fulfilled is $d_i^* = d_i - \sum_{j=1}^{m} B_{i,j} x_j^*$. Thus, if we take $d^* = (d_1^*, \ldots, d_m^*)$, we have a residual instance $I^* = (W, H, w, h, d^*)$ (we may eliminate from I^* the items with no demand).

If some $x_i^* > 0$ for some $i \in \{1, \ldots, m\}$, part of the demand is fulfilled by the solution x^*. In this case, we return B and x, we let $I = I^*$ and start a new iteration. If $x_i^* = 0$ for all $i \in \{1, \ldots, m\}$, no part of the demand is fulfilled by x^*. We solve then the instance I^* with the algorithm M-HFF (Modified HFF) that corresponds to the algorithm HFF modified to consider demands for the items, see [19]. We present in what follows the algorithm CG (Algorithm 8.4) that implements the iterative procedure we have described.

Note that, in each iteration, either part of the demand is fulfilled or we go to step 4. Thus, after a finite number of iterations the demand will be met (part of it eventually in step 4). In fact, it is easy to prove that step 3.6 of the algorithm CG is executed at most m times (see [18]).

We observe that the algorithm M-HFF can be implemented to run in polynomial time, see [19]. As its asymptotic performance bound is at most 2.077 (see [11]), we may expect that using M-HFF we produce solutions of good quality.

Input: An instance I = (W, H, w, h, d) of the 2CS problem.
Output: A solution for I.

1 Execute the algorithm SimplexCG$_2$ with parameters W, H, w, h, d obtaining B and x.
2 For $i = 1$ to m do $x_i^* = \lfloor x_i \rfloor$.
3 If $x_i^* > 0$ for some i, $1 \leq i \leq m$, then
  3.1 Return B and $x_1^*, \ldots, x_m^*$ (but do not halt).
  3.2 For $i = 1$ to m do
    3.2.1 For $j = 1$ to m do $d_i = d_i - B_{i,j} x_j^*$.
  3.3 Let $m' = 0$, $w' = (\,)$, $h' = (\,)$ and $d' = (\,)$.
  3.4 For $i = 1$ to m do
    3.4.1 If $d_i > 0$ then $m' = m' + 1$, $w' = w' \| (w_i)$, $h' = h' \| (h_i)$ and $d' = d' \| (d_i)$.
  3.5 If $m' = 0$ then halt.
  3.6 Let $m = m'$, $w = w'$, $h = h'$, $d = d'$ and go to step 1.
4 Return the solution of algorithm M-HFF executed with parameters W, H, w, h, d.

Algorithm 8.4: CG

We note that the algorithm CG can be used to solve the variant of 2CS, called BP^r, in which

orthogonal rotations of the items are allowed. For that, before we call the algorithm SDP, in step 3 of SimplexCG$_2$, it suffices to make the transformation explained at the end of Section 8.2.2. We will call SimplexCG$_2^r$ the variant of SimplexCG$_2$ with this transformation. It should be noted however that the algorithm M-HFF, called in step 6 of CG, does not use the fact that the items can be rotated.

We use a simple algorithm for the variant of BP^r in which all items have demand 1. This algorithm, called *First Fit Decreasing Height using Rotations* (FFDHR), has asymptotic approximation bound at most 4, as have been shown by Cintra in [18]. Substituting the call to M-HFF with a call to FFDHR, we obtain the algorithm CGR, that is a specialized version of CG for the BP^r problem.

We also tested another modification of the algorithm CG (and of CGR). This is the following: when we solve an instance, and the solution returned by SimplexCG$_2$ rounded down is equal to zero, instead of simply submitting this instance to M-HFF (or FFDHR), we use M-HFF (or FFDHR) to obtain a *good* pattern, and update the demands; if there is some item for which the demand is not fulfilled, we go to step 1. The *good* pattern used is the one with the largest occupated area of the bin.

Note that, the basic idea is to *perturb* the residual instances whose relaxed LP solution, rounded down, is equal to zero. With this procedure, it is expected that the solution obtained by SimplexCG$_2$ for the residual instance has more variables with value greater than 1. The algorithm CG^p, described in what follows (Algorithm 8.5), incorporates this modification.

Input: An instance $I = (W, H, w, h, d)$ of 2CS.
Output: A solution for I.

1 Execute the algorithm SimplexCG$_2$ with parameters W, H, w, h, d obtaining B and x.
2 For $i = 1$ to m do $x_i^* = \lfloor x_i \rfloor$.
3 If $x_i^* > 0$ for some i, $1 \leq i \leq m$, then
  3.1 Return B and $x_1^*, \ldots, x_m^*$ (but do not halt).
  3.2 For $i = 1$ to m do
    3.2.1 For $j = 1$ to m do $d_i = d_i - B_{i,j} x_j^*$.
  3.3 Let $m' = 0$, $w' = (\,)$, $h' = (\,)$ and $d' = (\,)$.
  3.4 For $i = 1$ to m do
    3.4.1 If $d_i > 0$ then $m' = m' + 1$, $w' = w' \| (w_i)$, $h' = h' \| (h_i)$ and $d' = d' \| (d_i)$.
  3.5 If $m' = 0$ then halt.
  3.6 Let $m = m'$, $w = w'$, $h = h'$, $d = d'$ and go to step 1.
4 Return a pattern generated by the algorithm M-HFF, executed with parameters W, H, w, h, d, that has the smallest wasted area, and update the demands.
5 If there are demands to be fulfilled, go to step 1.

Algorithm 8.5: CG^p

It should be noted that with this modification we cannot guarantee anymore that we have to make at most $m + 1$ calls to SimplexCG$_2$. It is however, easy to see that the algorithm CG^p in fact halts, as each time step 1 is executed, the demand decreases strictly. After a finite number of iterations the demand will be fulfilled and the algorithm halts.

8.4 The 2CS problem with bins of different sizes

In this section we adapt the algorithm CG for the BPV problem. Let $I = (W, H, V, w, h, d)$ be an instance of the BPV, where $W = (W_1, \ldots, W_b)$, $H = (H_1, \ldots, H_b)$ and $V = (V_1, \ldots, V_b)$ are lists of size b indicating the height, width, and value of each bin type i, $1 \leq i \leq b$. We can also represent each pattern j of the instance I as a vector p_j, whose i-th entry indicates the number of times item i occurs in this pattern. The BPV problem consists then in deciding how many times each pattern has to be used to meet the demands and minimize the total value of the bins that are used. Let n be the number of all possible patterns for I, and let P denote an $m \times n$ matrix whose columns are the patterns $p_1, \ldots, p_n$. If we denote by d the vector of the demands, then the following is an ILP formulation for the BPV problem: minimize $\sum_{j=1}^{n} V_j x_j$ subject to $Px = d$ and $x_j \geq 0$ and x_j integer for $j = 1, \ldots, n$. (The variable x_j indicates how many times pattern j is selected and V_j is the value of the bin type used in pattern j). The following is the corresponding relaxed formulation.

$$\begin{array}{ll} \text{minimize} & V_1 x_1 + \ldots + V_n x_n \\ \text{subject to} & Px = d \\ & x_j \geq 0 \quad j = 1, \ldots, n. \end{array} \tag{8.2}$$

In this case, we can also use the algorithm SDP to produce guillotine patterns. If y_i is the dual value corresponding to each item i, $1 \leq i \leq m$, then we want a pattern that maximizes $\sum_{i=1}^{m} y_i z_i$, where z_i is the number of times item i is used in the pattern. But in this case we have to solve the SDP problem to each possible bin size j, and a column j enters in the basis if $\sum_{i=1}^{m} y_i z_i > V_j$.

The algorithms of this section are an extension of the algorithms of the previous section. A more detailed description of these algorithms was done by Cintra [18].

We describe in the sequel the algorithm SimplexCG$_3$ that solves (8.2) (Algorithm 8.6). In this algorithm, we have a vector f of size m that indicates the bin associated with each column of the matrix B. This way, we can reconstruct a solution considering the vector f, and the entries of B, $guillotine$ and $position$. In the algorithm SimplexCG$_3$ we used subroutine SDP to solve the RK problem.

Input: An instance $I = (W, H, V, w, h, d)$ of the BPV problem.
Output: An optimal solution for (8.2), where the columns of P are the patterns for I.
Subroutine: The algorithm SDP for the RK problem.

1 Let f be a vector of size m where f_i is the smallest index of j such that $w_i \leq W_j$ and $h_i \leq H_j$.
2 Let $x = d$ and B be the identity matrix of order m.
3 Solve $y^T B = V_B^T$.
4 For $i = 1$ to b do
  4.1 Generate a new column z executing the algorithm SDP with parameters W_i, H_i, w, h, y.
  4.2 If $y^T z > V_i$, go to step 6.
5 Return B, f and x and halt (x corresponds to the columns of B).
6 Solve $Bw = z$.
7 Let $t = \min\,(\frac{x_j}{w_j} \mid 1 \leq j \leq m,\ w_j > 0)$.
8 Let $s = \min\,(j \mid 1 \leq j \leq m,\ \frac{x_j}{w_j} = t)$.
9 Let $f_s = i$
10 For $i = 1$ to m do
  10.1 $B_{i,s} = z_i$.
  10.2 If $i = s$ then $x_i = t$; otherwise, $x_i = x_i - w_i t$.
11 Go to step 3.

Algorithm 8.6: SimplexCG$_3$

The algorithm CGV (Algorithm 8.7) that solves the BPV problem using the algorithm SimplexCG$_3$ is very similar to the algorithm CG of Section 8.3, and therefore we omit the details.

We also considered the variants of the algorithm CGV, when we may have orthogonal rotations, and when the residual instance is solved with a *perturbation* method. In the latter case, to generate a pattern we use a bin for which the fraction $\frac{V_i}{H_i W_i}$ (for $i = 1, \ldots, b$) attains the minimum value.

8.5 The SP problem and the column generation method

The strip packing problem is mostly considered in the literature for the special case in which the demands are all equal to 1. Many approximation algorithms have been proposed for this problem. Coffman, Garey, Johnson and Tarjan [21] presented the algorithms NFDH and FFDH for the oriented case with asymptotic performance bounds 2 and 1.7, respectively. Algorithms with better performance bounds were obtained by Baker, Brown and Katseff [4] and also by Kenyon and Rémila [34]: $5/4$ and $(1 + \epsilon)$. Recently, a PTAS for the SP problem with rotations was obtained by Jansen and van Stee [32]. In 2005, Seiden and Woeginger [45] presented an

Input: An instance I = (W, H, V, w, h, d) of BPV.
Output: A solution for I.

1 Execute the algorithm SimplexCG$_2$ with parameters B, w, h, d obtaining B, b and x.
2 For $i = 1$ to m do $x_i^* = \lfloor x_i \rfloor$.
3 If $x_i^* > 0$ for some i, $1 \leq i \leq m$, then
  3.1 Return B, b and $x_1^*, \ldots, x_m^*$ (but do not halt).
  3.2 For $i = 1$ to m do
    3.2.1 For $j = 1$ to m do $d_i = d_i - B_{i,j}x_j^*$.
  3.3 Let $m' = 0$, $h' = (\ )$, $w' = (\ )$ and $d' = (\ )$.
  3.4 For $i = 1$ to m do
    3.4.1 If $d_i > 0$ then $m' = m' + 1$, $w' = w' \| (w_i)$, $h' = h' \| (h_i)$ and $d' = d' \| (d_i)$.
  3.5 If $m' = 0$ then halt.
  3.6 Let $m = m'$, $w = w'$, $h = h'$, $d = d'$ and go to step 1.
4 Let $V^* = \min\left(\frac{V_i}{H_i W_i} \mid i = 1, \ldots, b\right)$ and $j = \min\left(i \mid \frac{V_i}{H_i W_i} = V^*\right)$.
5 Return the solution of algorithm M-HFF executed with parameters W_j, H_j, w, h, d.

Algorithm 8.7: CGV

analysis of the quality of a k-stage guillotine strip packing versus a globally optimum packing. They showed that for $k = 2$ no algorithm can guarantee any bounded asymptotic performance ratio. When $k = 3$ (resp. $k = 4$) an asymptotic performance ratio arbitrarily close to 1.69103 (resp. 1) can be obtained. Although some of the approximation algorithms above have bounds very close to 1, most of these results are more of theoretical relevance. Other approaches include genetic algorithms [40], branch and bound and integer linear programming models [35, 37].

All algorithms for the SP problem mentioned above consider that each item has demand 1. Although the column generation approach can be easily applied to the problem SP, it is less investigated under this approach. One of the main advantages of this approach is the possibility to consider larger values of demands, as this case has many industrial applications.

Let $I = (W, H, w, h, d)$ be an instance of the SP problem. We consider that the first cut stage is done in the horizontal direction of the strip; furthermore, two subsequent cuts must be at a distance at most H. We call *H-pattern* a pattern corresponding to a packing between two subsequent horizontal cuts (that has to be at a maximum distance H).

Let $p_1, p_2, \ldots, p_n$ be the set of all possible H-patterns. Denote by H_i the height of the H-pattern p_i and let P be the matrix whose columns are the patterns $p_1, p_2, \ldots, p_n$. In this case, the following is an ILP formulation for the SP problem: minimize $\sum_{j=1}^{n} H_j x_j$ subject to $Px = d$ and $x_j \geq 0$ and x_j integer for $j = 1, \ldots, n$. To solve this ILP we can use the same approach we used for the problem BPV. In fact, we can reduce the SP problem to the BPV problem. For that, note that each H-pattern with height H_i corresponds to a bin with dimensions (W, H_i) and

value precisely H_i.

Let $Q = \{q_1, \ldots, q_s\}$ be the set of all discretization points of the height H (this will be the maximum height of the bins).

For 2-staged cutting patterns, we can consider H as the maximum height of an item, that is, $H = \max(h_1, \ldots, h_m)$. In this case, Q is the set of the heights of the items. If there are s different heights, we have H-patterns (bins) with width $W_i = W$ and height H_i, for $1 \leq i \leq s$.

The algorithm we propose to solve the SP problem, called CGS, uses basically the algorithm CGV with two modifications. First, the residual instance is solved with the algorithm FFDH. Second, every call to the algorithm SimplexCG$_3$ solves only one instance of the RK problem, consisting of a knapsack of size (W, H).

We note that, looking at the entries of V, $guillotine$ and $position$ produced by algorithm SDP (algorithm for the staged RK problem) we can obtain solutions for each height in Q: we just have to access positions (W, h_i) of these variables, for each $h_i \in Q$, $1 \leq i \leq s$. This last modification is very important, as s can be very large and solving instances of the RK problem for each of the s different bins would consume a lot of time. We did not use this idea for the BPV problem since it is not always better to solve only instances of RK with the largest bin dimensions.

Note that, in the BPV problem, the instances may consist of bins of different widths. Consider, for example, an instance consisting of x bins: one bin with size (r, r^2), another one with size (r^2, r) and some other $x - 2$ bins with dimensions smaller than r, for some integer r. If we call the algorithm DP for a bin of dimensions (r^2, r^2) and assume that the number of discretization points is linearly proportional to the dimensions of the bin, then the algorithm will consume time $O(kr^6)$. But if we solve for each of the bins, the algorithm will consume time $O(kxr^5)$.

The reader should note that the first cutting phase is done automatically by the column generation algorithm by choosing the best bins in a solution. Therefore, the algorithm SDP is called with the first cutting phase in the vertical direction and one cutting phase less than the number of stages of the instance.

We implemented the algorithm CGS and its variant CGS^r (for the orthogonal rotation case) and CGS^p with a perturbed residual instance. In the algorithm CGS^p a good way to perturb the instance is to generate a level by the algorithm FFDH with minimum wasted area (considering the height of the level). When rotations are allowed we use an algorithm, which we denote by FFDHR2, to generate a perturbed instance. This algorithm works like the algorithm FFDH, but if an item cannot be packed in any of the existing levels then the algorithm tries to pack it in the other orientation before creating a new level.

8.6 Computational results

In the next subsections we present the computational results obtained with the implementations of the algorithms we have described. All algorithms were implemented in C language. The computational tests were run on a computer with processor Intel Pentium IV, clock of 1.8GHz, memory of 512Mb and operating system *Linux* using the LP solver CLP (COIN-OR Linear Program Solver) [22].

For all problems we have performed computational tests considering staged guillotine patterns with and without orthogonal rotations. Following other papers in the literature (see [12, 13, 46]), we assume that the first cutting stage is performed in the horizontal direction.

8.6.1 Computational results for the RK problem

The performance of the algorithm SDP was tested with the instances of RK available in the OR-LIBRARY[1] (see Beasley [7] for a brief description of this library). We considered the 13 instances of RK, called *gcut1, . . .,gcut13* available in this library. For all these instances, with exception of instance *gcut13*, optimal solutions had already been found [5]. We considered the the SDP algorithm with the number of stages $k \in \{2, 3, 4\}$.

In [20], Cintra and Wakabayashi already found an optimal solution for instance *gcut13*, but considering non-staged patterns. Using the algorithm SDP we found optimal solutions for the instance *gcut13* for the staged patterns considered. We notice that the solution found with 3-staged patterns already corresponds to an optimal solution without restriction on the number of stages. This solution was found in less than 22 seconds and is shown in Figure 8.2. In all these instances the value of each item is precisely its area. Caprara and Monaci [14] and Fekete and Schepers [26] could not find an optimal solution for this instance in 1800 seconds in recent machines (a Pentium III 800MHz and Pentium IV 2.8GHz with 1Gb of memory, respectively). We recall that their approaches are for the more general setting in which the cuts need not be guillotine. We note that, in this general case, our approach can be used to obtain a lower bound.

Since the algorithm solves all instances of the OR-LIBRARY in a few seconds, we construct other four instances (*gcut14 – gcut17*) based on the available instances. We join the items instance *gcut13* with the items of instances *gcut9, gcut10, gcut11* and *gcu12*, obtaining the new instances. For each one of these new instances we considered a knapsack with size (3500,3500). In Table 8.1 we give some information about the instances.

The computational results are shown in Table 8.2. The column "Waste" shows —for each solution found— the percentage of the area of the bin that does not correspond to any item. The column "Time" indicates the time required to solve the instance; the entry 0 indicates that the

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

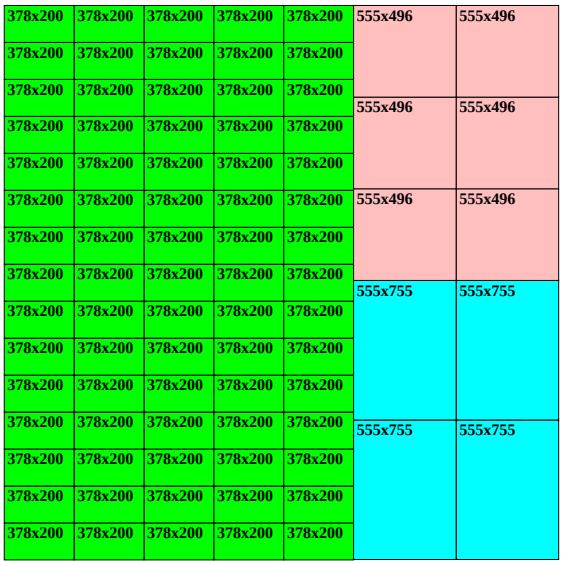

Figure 8.2: The optimal solution for *gcut13* found by the algorithm SDP with 3-staged patterns. The small squares have dimensions $(378, 200)$ and the squares in the bottom have dimensions $(555, 496)$ and $(555, 755)$.

time required is less than 0.000001 seconds. On the average, the waste for 2-staged patterns was less than 1% larger than the waste for 4-staged patterns. The space utilization comparing 3- and 4-staged patterns are very close. The solutions that differs in waste are the solutions of instances *gcut8, gcut14, gcut15, gcut16* and *gcut17*. Moreover, all solutions found with 4-staged patterns of instances *gcut1* through *gcut13* also correspond to optimal solutions for the unrestricted case as one can compare to the results in [20].

To run tests for the case in which orthogonal rotations are allowed, we considered the instances *gcut1,. . .,gcut17*, and named them correspondingly as *gcut1r,. . .,gcut17r* (meaning that rotations are allowed). The performance algorithm SDP for these instances is presented in table 8.3. We remark that, comparing with the problem without rotations, for some instances the time increased and the waste decreased (on the average less than 1%), as one would expect.

For these instances, we can also note that the solutions with 4-staged patterns correspond to optimal solutions for the unrestricted case (see [20]) except for instance *gcut14r*. The differences on space utilization from the 3-staged to the 4-staged patterns are also very small, differing in the instances *gcut3r*, *gcut8r*, *gcut14r*, *gcut15r* and *gcut16r*.

Instance	Quantity of items	Dimensions of the bin	r	s
gcut1	10	(250, 250)	28	9
gcut2	20	(250, 250)	39	52
gcut3	30	(250, 250)	81	42
gcut4	50	(250, 250)	85	84
gcut5	10	(500, 500)	19	27
gcut6	20	(500, 500)	34	42
gcut7	30	(500, 500)	66	33
gcut8	50	(500, 500)	97	136
gcut9	10	(1000, 1000)	31	11
gcut10	20	(1000, 1000)	29	55
gcut11	30	(1000, 1000)	69	109
gcut12	50	(1000, 1000)	155	124
gcut13	32	(3000, 3000)	1457	2310
gcut14	42	(3500, 3500)	2390	2861
gcut15	52	(3500, 3500)	2422	2933
gcut16	62	(3500, 3500)	2559	2943
gcut17	82	(3500, 3500)	2676	2953

Table 8.1: Instances information.

	Quant.		2-staged			3-staged			4-staged		
Inst.	of items	Dimensions of the bin	Optimal Solution	Waste	Time (sec)	Optimal Solution	Waste	Time (sec)	Optimal Solution	Waste	Time (sec)
gcut1	10	(250, 250)	56460	9.66%	0	56460	9.66%	0	56460	9.66%	0
gcut2	20	(250, 250)	60076	3.878%	0	60536	3.142%	0	60536	3.142%	0
gcut3	30	(250, 250)	60133	3.787%	0	61036	2.342%	0	61036	2.342%	0
gcut4	50	(250, 250)	61698	1.283%	0	61698	1.283%	0.01	61698	1.283%	0
gcut5	10	(500, 500)	246000	1.600%	0	246000	1.600%	0	246000	1.600%	0
gcut6	20	(500, 500)	235058	5.977%	0	238998	4.401%	0	238998	4.401%	0
gcut7	30	(500, 500)	242567	2.973%	0	242567	2.973%	0	242567	2.973%	0.017
gcut8	50	(500, 500)	245758	1.697%	0	245758	1.697%	0	246633	1.347%	0.071
gcut9	10	(1000, 1000)	971100	2.890%	0	971100	2.890%	0	971100	2.890%	0
gcut10	20	(1000, 1000)	982025	1.798%	0	982025	1.798%	0	982025	1.798%	0
gcut11	30	(1000, 1000)	974638	2.536%	0	980096	1.990%	0	980096	1.990%	0
gcut12	50	(1000, 1000)	977768	2.223%	0.01	979986	2.001%	0	979986	2.001%	0.01
gcut13	32	(3000, 3000)	8906216	1.042%	21.82	8997780	0.025%	32.98	8997780	0.025%	43.72
gcut14	42	(3500, 3500)	12216788	0.271%	124.55	12239634	0.085%	175.96	12242100	0.064%	264.41
gcut15	52	(3500, 3500)	12215614	0.281%	137.21	12239904	0.082%	189.53	12242100	0.064%	289.98
gcut16	62	(3500, 3500)	12210837	0.320%	177.21	12243100	0.056%	239.24	12244511	0.045%	371.60
gcut17	82	(3500, 3500)	12232948	0.139%	223.13	12246422	0.029%	290.07	12246694	0.027%	456.00

Table 8.2: Performance of the algorithm SDP for 2-, 3- and 4-staged patterns.

	Qty.		2-staged			3-staged			4-staged		
Inst.	of items	Dimensions of the bin	Optimal Solution	% Waste	Time (sec)	Optimal Solution	% Waste	Time (sec)	Optimal Solution	% Waste	Time (sec)
gcut1r	10	(250, 250)	58136	6.982	0	58136	6.982	0	58136	6.982	0
gcut2r	20	(250, 250)	60611	3.022	0	60611	3.022	0	60611	3.022	0
gcut3r	30	(250, 250)	60485	3.224	0	61399	1.762	0	61626	1.398	0
gcut4r	50	(250, 250)	62265	0.376	0.01	62265	0.376	0.01	62265	0.376	0.01
gcut5r	10	(500, 500)	246000	1.600	0	246000	1.600	0	246000	1.600	0
gcut6r	20	(500, 500)	240951	3.620	0	240951	3.620	0	240951	3.620	0
gcut7r	30	(500, 500)	245866	1.654	0.01	245886	1.654	0.01	245866	1.654	0
gcut8r	50	(500, 500)	247260	1.096	0.01	247462	1.015	0.02	247787	0.885	0.02
gcut9r	10	(1000, 1000)	971100	2.890	0	971100	2.890	0	971100	2.890	0
gcut10r	20	(1000, 1000)	982025	1.798	0	982025	1.798	0	982025	1.798	0
gcut11r	30	(1000, 1000)	980096	1.990	0.02	980096	1.990	0.03	980096	1.990	0.04
gcut12r	50	(1000, 1000)	988694	1.131	0.03	988694	1.131	0.05	988694	1.131	0.06
gcut13r	32	(3000, 3000)	8997780	0.025	106.3	9000000	0.0	129.64	9000000	0.0	226.75
gcut14r	42	(3500, 3500)	12240515	0.077%	322.77	12247700	0.019%	418.26	12247796	0.018%	702.19
gcut15r	52	(3500, 3500)	12242904	0.058%	337.27	12248176	0.015%	437.72	12250000	0.000%	725.21
gcut16r	62	(3500, 3500)	12243100	0.056%	368.20	12249625	0.003%	465.92	12250000	0.000%	800.55
gcut17r	82	(3500, 3500)	12242998	0.057%	393.52	12250000	0.000%	495.39	12250000	0.000%	829.92

Table 8.3: Performance of the algorithm SDP for 2-, 3- and 4-staged patterns with rotations.

8.6.2 Computational results for the 2CS problem

We did not find instances for the 2CS problem in the OR-LIBRARY. We tested the algorithms CG and CG^p with the instances *gcut1, ..., gcut12*, associating with each item i a randomly generated demand d_i between 1 and 100 (varying demands). We called these instances *gcut1d, ..., gcut12d*.

In the tables of the results, LB denotes the lower bound (given by the rounded up solution of (8.1)) for the value of an optimal integer solution.

We used the algorithm CG^p with subroutine SDP and $k = 2, 3, 4$.

The tests for this case are presented in tables 8.4–8.9. For all tests, the algorithms CG and CG^p obtained solutions in a small amount of time.

Instance	Solution of CG	LB	Difference from LB	Time (sec)	Columns Generated	Solution of M-HFF	Improvement over M-HFF
gcut1d	295	295.0	0.000%	0.03	19	322	8.39%
gcut2d	345	345.0	0.000%	0.37	137	360	4.17%
gcut3d	343	342.0	0.292%	1.10	388	374	8.29%
gcut4d	845	845.0	0.000%	4.02	828	878	3.76%
gcut5d	207	207.0	0.000%	0.03	19	224	7.59%
gcut6d	375	375.0	0.000%	0.14	77	395	5.06%
gcut7d	600	600.0	0.000%	0.59	278	642	6.54%
gcut8d	720	720.0	0.000%	3.35	592	765	5.88%
gcut9d	135	135.0	0.000%	0.07	48	141	4.26%
gcut10d	315	315.0	0.000%	0.14	79	328	3.96%
gcut11d	349	349.0	0.000%	0.68	224	375	6.93%
gcut12d	676	675.0	0.148%	4.03	660	722	6.37%

Table 8.4: Performance of the algorithm CG with 2-staged patterns.

For the 2-staged cutting, the algorithms CG and CG^p obtained optimum solutions for all instances, except for two of them (on the average, the difference from LB was 0.036%). When compared to the solution of M-HFF, the improvement was 5.93% on the average. This is a great improvement, since M-HFF is also restricted to 2-staged patterns.

For the 3-staged problem, algorithm CG^p, found one more optimal solution (*gcut10d*) comparing to the results of algorithm CG. For the 4-staged case the algorithm CG found a better solution to instance *gcutd7d* than the one found by the algorithm CG^p. On the other hand the algorithm CG^p found an optimal solution to instance *gcut11d* while CG does not.

The algorithms had a good performance both in terms of the quality of the solution and in terms of the time required. The improvement of the algorithm CG^p for the 4-staged case, over M-HFF was, on the average, 8.89%, for example.

Instance	Solution of CG	LB	Difference from LB	Time (sec)	Columns Generated	Solution of M-HFF	Improvement over M-HFF
gcut1d	295	295.0	0.000%	0.03	21	322	8.39%
gcut2d	345	345.0	0.000%	0.45	173	360	4.17%
gcut3d	343	342.0	0.292%	1.31	534	374	8.29%
gcut4d	845	845.0	0.000%	5.99	1506	878	3.76%
gcut5d	207	207.0	0.000%	0.03	21	224	7.59%
gcut6d	375	375.0	0.000%	0.16	86	395	5.06%
gcut7d	600	600.0	0.000%	0.71	357	642	6.54%
gcut8d	720	720.0	0.000%	3.50	693	765	5.88%
gcut9d	135	135.0	0.000%	0.08	57	141	4.26%
gcut10d	315	315.0	0.000%	0.21	122	328	3.96%
gcut11d	349	349.0	0.000%	0.79	289	375	6.93%
gcut12d	676	675.0	0.148%	5.28	1167	722	6.37%

Table 8.5: Performance of the algorithm CG^p with 2-staged patterns.

We also tested the algorithms with rotations on the instances *gcut1dr,. . ., gcut12dr*. See tables 8.10–8.15.

Notice that the algorithm CGR^p obtains better solutions than the algorithm CGR in several instances. For the algorithm CGR^p, in the 2-staged case, the difference from the lower bound was 0.220%, on the average and the improvement over the FFDHR algorithm was 10.14%. These numbers are very close to the ones we can obtain for the 3- and 4-staged version.

Instance	Solution of CG	LB	Difference from LB	Time (sec)	Columns Generated	Solution of M-HFF	Improvement over M-HFF
gcut1d	294	294.0	0.000%	0.04	24	322	8.70%
gcut2d	345	345.0	0.000%	0.34	101	360	4.17%
gcut3d	333	333.0	0.000%	0.87	285	374	10.96%
gcut4d	837	836.0	0.120%	5.44	1015	878	4.67%
gcut5d	198	197.0	0.508%	0.05	29	224	11.61%
gcut6d	344	343.0	0.292%	0.20	100	395	12.91%
gcut7d	591	591.0	0.000%	0.46	203	642	7.94%
gcut8d	692	690.0	0.290%	7.02	985	765	9.54%
gcut9d	132	131.0	0.763%	0.08	49	141	6.38%
gcut10d	294	293.0	0.341%	0.12	58	328	10.37%
gcut11d	331	330.0	0.303%	1.42	379	375	11.73%
gcut12d	673	672.0	0.149%	4.32	601	722	6.79%

Table 8.6: Performance of the algorithm CG with 3-staged patterns.

Instance	Solution of CG	LB	Difference from LB	Time (sec)	Columns Generated	Solution of M-HFF	Improvement over M-HFF
gcut1d	294	294.0	0.000%	0.04	26	322	8.70%
gcut2d	345	345.0	0.000%	0.38	135	360	4.17%
gcut3d	333	333.0	0.000%	1.30	506	374	10.96%
gcut4d	837	836.0	0.120%	8.19	1878	878	4.67%
gcut5d	198	197.0	0.508%	0.06	41	224	11.61%
gcut6d	344	343.0	0.292%	0.22	113	395	12.91%
gcut7d	591	591.0	0.000%	0.52	229	642	7.94%
gcut8d	692	690.0	0.290%	8.88	1563	765	9.54%
gcut9d	132	131.0	0.763%	0.10	70	141	6.38%
gcut10d	293	293.0	0.000%	0.13	73	328	10.67%
gcut11d	331	330.0	0.303%	2.28	710	375	11.73%
gcut12d	673	672.0	0.149%	4.95	885	722	6.79%

Table 8.7: Performance of the algorithm CG^p with 3-staged patterns.

Instance	Solution of CG	LB	Difference from LB	Time (sec)	Columns Generated	Solution of M-HFF	Improvement over M-HFF
gcut1d	294	294.0	0.000%	0.04	23	322	8.70%
gcut2d	345	345.0	0.000%	0.46	133	360	4.17%
gcut3d	332	332.0	0.000%	0.87	260	374	11.23%
gcut4d	837	836.0	0.120%	4.27	668	878	4.67%
gcut5d	198	197.0	0.508%	0.05	28	224	11.61%
gcut6d	344	343.0	0.292%	0.19	98	395	12.91%
gcut7d	592	591.0	0.169%	0.27	103	642	7.79%
gcut8d	691	690.0	0.145%	9.68	1247	765	9.67%
gcut9d	131	131.0	0.000%	0.05	35	141	7.09%
gcut10d	294	293.0	0.341%	0.14	70	328	10.37%
gcut11d	331	330.0	0.303%	1.26	285	375	11.73%
gcut12d	673	672.0	0.149%	5.06	640	722	6.79%

Table 8.8: Performance of the algorithm CG with 4-staged patterns.

Instance	Solution of CG	LB	Difference from LB	Time (sec)	Columns Generated	Solution of M-HFF	Improvement over M-HFF
gcut1d	294	294.0	0.000%	0.04	25	322	8.70%
gcut2d	345	345.0	0.000%	0.49	157	360	4.17%
gcut3d	332	332.0	0.000%	1.76	621	374	11.23%
gcut4d	837	836.0	0.120%	7.27	1606	878	4.67%
gcut5d	198	197.0	0.508%	0.06	40	224	11.61%
gcut6d	344	343.0	0.292%	0.26	136	395	12.91%
gcut7d	593	591.0	0.338%	0.61	295	642	7.63%
gcut8d	691	690.0	0.145%	10.33	1539	765	9.67%
gcut9d	131	131.0	0.000%	0.08	55	141	7.09%
gcut10d	294	293.0	0.341%	0.17	93	328	10.37%
gcut11d	330	330.0	0.000%	1.88	535	375	12.00%
gcut12d	673	672.0	0.149%	5.70	927	722	6.79%

Table 8.9: Performance of the algorithm CG^p with 4-staged patterns.

Instance	Solution of CG	LB	Difference from LB	Time (sec)	Columns Generated	Solution of M-HFF	Improvement over M-HFF
gcut1dr	291	291.0	0.000%	0.03	21	291	0.00%
gcut2dr	283	282.0	0.355%	3.01	263	314	9.87%
gcut3dr	318	316.0	0.633%	2.83	580	347	8.36%
gcut4dr	837	836.0	0.120%	5.50	722	846	1.06%
gcut5dr	175	175.0	0.000%	0.07	33	198	11.62%
gcut6dr	302	302.0	0.000%	0.44	156	371	18.60%
gcut7dr	543	542.0	0.185%	0.69	178	623	12.84%
gcut8dr	650	650.0	0.000%	6.76	602	734	11.44%
gcut9dr	126	125.0	0.800%	0.07	38	143	11.89%
gcut10dr	271	270.0	0.370%	0.37	128	301	9.97%
gcut11dr	300	299.0	0.334%	6.15	388	342	12.28%
gcut12dr	602	601.0	0.166%	21.55	835	696	13.51%

Table 8.10: Performance of the algorithm CGR with rotations and 2-staged patterns.

Instance	Solution of CGR^p	LB	Difference from LB	Time (sec)	Columns Generated	Solution of FFDHR	Improvement over FFDHR
gcut1dr	291	291.0	0.000%	0.05	26	291	0.00%
gcut2dr	283	282.0	0.355%	3.69	359	314	9.87%
gcut3dr	317	316.0	0.316%	4.35	1023	347	8.65%
gcut4dr	837	836.0	0.120%	9.47	1523	846	1.06%
gcut5dr	175	175.0	0.000%	0.09	45	198	11.62%
gcut6dr	302	302.0	0.000%	0.45	166	371	18.60%
gcut7dr	543	542.0	0.185%	0.72	193	623	12.84%
gcut8dr	650	650.0	0.000%	6.85	630	734	11.44%
gcut9dr	126	125.0	0.800%	0.10	61	143	11.89%
gcut10dr	271	270.0	0.370%	0.45	177	301	9.97%
gcut11dr	300	299.0	0.334%	8.32	677	342	12.28%
gcut12dr	602	601.0	0.166%	24.55	1207	696	13.51%

Table 8.11: Performance of the algorithm CGR^p with rotations and 2-staged patterns.

Instance	Solution of CGR^p	LB	Difference from LB	Time (sec)	Columns Generated	Solution of FFDHR	Improvement over FFDHR
gcut1dr	291	291.0	0.000%	0.04	22	291	0.00%
gcut2dr	283	282.0	0.355%	3.50	256	314	9.87%
gcut3dr	315	313.0	0.639%	3.28	585	347	9.22%
gcut4dr	836	836.0	0.000%	6.62	782	846	1.18%
gcut5dr	175	174.0	0.575%	0.10	48	198	11.62%
gcut6dr	302	301.0	0.332%	0.78	228	371	18.60%
gcut7dr	544	542.0	0.369%	1.63	350	623	12.68%
gcut8dr	651	650.0	0.154%	11.92	716	734	11.31%
gcut9dr	123	122.0	0.820%	0.08	39	143	13.99%
gcut10dr	270	270.0	0.000%	0.35	89	301	10.30%
gcut11dr	299	298.0	0.336%	5.99	321	342	12.57%
gcut12dr	603	601.0	0.333%	35.15	976	696	13.36%

Table 8.12: Performance of the algorithm CGR with rotations and 3-staged patterns.

Instance	Solution of CGR^p	LB	Difference from LB	Time (sec)	Columns Generated	Solution of FFDHR	Improvement over FFDHR
gcut1dr	291	291.0	0.000%	0.05	27	291	0.00%
gcut2dr	283	282.0	0.355%	4.24	331	314	9.87%
gcut3dr	315	313.0	0.639%	5.13	1056	347	9.22%
gcut4dr	836	836.0	0.000%	9.67	1344	846	1.18%
gcut5dr	175	174.0	0.575%	0.14	69	198	11.62%
gcut6dr	301	301.0	0.000%	1.04	330	371	18.87%
gcut7dr	543	542.0	0.185%	2.10	509	623	12.84%
gcut8dr	651	650.0	0.154%	13.57	967	734	11.31%
gcut9dr	123	122.0	0.820%	0.10	53	143	13.99%
gcut10dr	270	270.0	0.000%	0.35	92	301	10.30%
gcut11dr	299	298.0	0.336%	6.41	417	342	12.57%
gcut12dr	602	601.0	0.166%	35.66	1142	696	13.51%

Table 8.13: Performance of the algorithm CGR^p with rotations and 3-staged patterns.

Instance	Solution of CGR^p	LB	Difference from LB	Time (sec)	Columns Generated	Solution of FFDHR	Improvement over FFDHR
gcut1dr	291	291.0	0.000%	0.05	21	291	0.00%
gcut2dr	283	282.0	0.355%	1.95	174	314	9.87%
gcut3dr	315	313.0	0.639%	3.05	439	347	9.22%
gcut4dr	836	836.0	0.000%	6.41	583	846	1.18%
gcut5dr	175	174.0	0.575%	0.13	53	198	11.62%
gcut6dr	302	301.0	0.332%	0.54	147	371	18.60%
gcut7dr	543	542.0	0.185%	1.87	348	623	12.84%
gcut8dr	652	650.0	0.308%	14.51	691	734	11.17%
gcut9dr	123	122.0	0.820%	0.09	42	143	13.99%
gcut10dr	270	270.0	0.000%	0.45	103	301	10.30%
gcut11dr	299	298.0	0.336%	11.84	386	342	12.57%
gcut12dr	603	601.0	0.333%	40.92	903	696	13.36%

Table 8.14: Performance of the algorithm CGR with rotations and 4-staged patterns.

Instance	Solution of CGR^p	LB	Difference from LB	Time (sec)	Columns Generated	Solution of FFDHR	Improvement over FFDHR
gcut1dr	291	291.0	0.000%	0.05	26	291	0.00%
gcut2dr	283	282.0	0.355%	2.22	274	314	9.87%
gcut3dr	314	313.0	0.319%	6.85	1103	347	9.51%
gcut4dr	836	836.0	0.000%	12.81	1446	846	1.18%
gcut5dr	175	174.0	0.575%	0.14	65	198	11.62%
gcut6dr	302	301.0	0.332%	0.74	230	371	18.60%
gcut7dr	542	542.0	0.000%	2.46	568	623	13.00%
gcut8dr	651	650.0	0.154%	18.35	1159	734	11.31%
gcut9dr	123	122.0	0.820%	0.11	58	143	13.99%
gcut10dr	270	270.0	0.000%	0.44	109	301	10.30%
gcut11dr	299	298.0	0.336%	20.08	996	342	12.57%
gcut12dr	602	601.0	0.166%	47.96	1535	696	13.51%

Table 8.15: Performance of the algorithm CGR^p with rotations and 4-staged patterns.

8.6.3 Computational results for the BPV problem

We have tested the algorithm CGV^p with the instances *gcut1d,. . ., gcut12d,* defining three different bins. For each bin in the original instances, we define two others. Given an instance, let (W, H) be the bin dimensions of this instance. In our modified instances, one bin has dimensions $(1.2W, 0.8H)$ and the other has dimensions $(1.1W, 0.9H)$. The value of each bin corresponds to its area $W \times H$.

For the k-staged version of he BPV problem, we present tests for the algorithm CGV^p with $k = 2, 3, 4$ (see tables 8.16–8.18). We do not present the results of the algorithm CGV since CGV^p got better results in several instances.

Instance	Solution of CGV^p	LB	Difference from LB	Time (sec)	Columns Generated
gcut1d	14880000	14822812.5	0.386%	0.58	397
gcut2d	16820625	16740781.3	0.477%	1.31	492
gcut3d	20267500	20149803.6	0.584%	21.83	7877
gcut4d	46591875	46523511.2	0.147%	60.56	11569
gcut5d	42022500	41667500.0	0.852%	0.17	110
gcut6d	78167500	77621562.5	0.703%	0.96	539
gcut7d	124257500	123946562.5	0.251%	2.90	1316
gcut8d	161575000	161074884.1	0.310%	23.67	3958
gcut9d	131830000	130802500.0	0.786%	0.12	86
gcut10d	262470000	260444166.7	0.778%	0.81	434
gcut11d	304440000	303137516.6	0.430%	18.58	6926
gcut12d	611230000	609519416.7	0.281%	36.65	5452

Table 8.16: Performance of the algorithm CGV^p with 2-staged patterns.

For $k = 2$ (resp. 3 and 4) the difference of the solution obtained by the algorithm from the lower bound was of 0.498% (resp. 0.505% and 0.437%) on the average. The algorithm CGV^p with $k = 3$ (resp. $k = 4$) has an increase of $84,56\%$ (resp. 143.68%) of computational time when compared with $k = 2$, on the average.

When orthogonal rotations are allowed in the BPV problem, we note that it becomes harder to solve the instances. We can see that the large instances require several minutes to be solved. See tables 8.19–8.21.

Instance	Solution of CGV^p	LB	Difference from LB	Time (sec)	Columns Generated
gcut1d	14880000	14822812.5	0.386%	0.45	310
gcut2d	15768125	15679972.9	0.562%	6.37	2587
gcut3d	19914375	19830115.7	0.425%	8.72	3086
gcut4d	46413750	46269759.9	0.311%	103.53	19002
gcut5d	41737500	41517500.0	0.530%	0.70	493
gcut6d	74440000	73967812.5	0.638%	3.96	2116
gcut7d	123135000	122531666.7	0.492%	9.16	3940
gcut8d	155612500	155267743.8	0.222%	91.62	15493
gcut9d	130730000	129600000.0	0.872%	0.18	119
gcut10d	254160000	252596666.7	0.619%	1.61	1030
gcut11d	295270000	292967500.0	0.786%	17.55	5627
gcut12d	603220000	601848214.3	0.228%	66.44	9763

Table 8.17: Performance of the algorithm CGV^p with 3-staged patterns.

Instance	Solution of CGVR^p	LB	Difference from LB	Time (sec)	Columns Generated
gcut1dr	13908750	13828125.0	0.583%	0.67	416
gcut2dr	15474375	15432371.3	0.272%	37.05	4616
gcut3dr	19436875	19310805.3	0.653%	45.33	12159
gcut4dr	44905000	44767392.4	0.307%	166.68	21902
gcut5dr	40382500	40087187.5	0.737%	0.74	341
gcut6dr	71162500	70839625.0	0.456%	5.49	2411
gcut7dr	115312500	114817716.3	0.431%	56.78	13326
gcut8dr	153410000	152634892.3	0.508%	394.05	28128
gcut9dr	121040000	119568000.0	1.231%	1.14	756
gcut10dr	249260000	247872857.1	0.560%	5.68	1545
gcut11dr	289430000	286973906.4	0.856%	290.64	23447
gcut12dr	564650000	562898801.3	0.311%	690.59	28565

Table 8.19: Performance of the algorithm CGVR^p with rotations and 2-staged patterns.

Instance	Solution of CGV^p	LB	Difference from LB	Time (sec)	Columns Generated
gcut1d	14880000	14822812.5	0.386%	0.44	307
gcut2d	15730625	15673933.2	0.362%	8.03	3163
gcut3d	19864375	19769831.3	0.478%	40.23	12327
gcut4d	46343750	46257603.4	0.186%	125.23	18410
gcut5d	41737500	41517500.0	0.530%	0.68	489
gcut6d	74187500	73967812.5	0.297%	2.27	1005
gcut7d	122745000	122295271.7	0.368%	9.09	3715
gcut8d	155832500	155221710.8	0.393%	117.61	17864
gcut9d	129360000	128389230.8	0.756%	1.01	727
gcut10d	254130000	252565036.2	0.620%	2.76	1649
gcut11d	294200000	292879166.7	0.451%	33.78	8413
gcut12d	602360000	599851250.0	0.418%	68.63	7886

Table 8.18: Performance of the algorithm CGV^p with 4-staged patterns.

Instance	Solution of CGVR^p	LB	Difference from LB	Time (sec)	Columns Generated
gcut1dr	13823750	13790625.0	0.240%	0.70	407
gcut2dr	15158750	15083409.1	0.499%	33.76	2676
gcut3dr	19235000	19120561.8	0.599%	45.67	8410
gcut4dr	44672500	44627391.4	0.101%	307.66	31450
gcut5dr	38887500	38456458.3	1.121%	2.42	1044
gcut6dr	70090000	69717232.1	0.535%	12.23	3892
gcut7dr	115220000	114605812.2	0.536%	52.29	10316
gcut8dr	151917500	151467609.8	0.297%	502.60	32303
gcut9dr	120290000	119104183.0	0.996%	0.41	210
gcut10dr	247580000	246552500.0	0.417%	5.55	2065
gcut11dr	283940000	282079863.6	0.659%	269.26	16320
gcut12dr	561610000	559820015.8	0.320%	1003.48	39675

Table 8.20: Performance of the algorithm CGVR^p with rotations and 3-staged patterns.

Instance	Solution of CGVR^p	LB	Difference from LB	Time (sec)	Columns Generated
gcut1dr	13823750	13790625.0	0.240%	0.83	415
gcut2dr	15161875	15083409.1	0.520%	46.73	3010
gcut3dr	19181875	19118423.5	0.332%	96.76	16255
gcut4dr	44723750	44575105.3	0.333%	253.43	27104
gcut5dr	38890000	38454765.6	1.132%	4.72	1662
gcut6dr	70192500	69599732.1	0.852%	10.45	2639
gcut7dr	114867500	114503487.9	0.318%	70.18	12339
gcut8dr	151745000	151462312.9	0.187%	605.13	30285
gcut9dr	119730000	118806666.7	0.777%	2.73	1198
gcut10dr	248620000	246552500.0	0.839%	9.25	3409
gcut11dr	283560000	281851974.2	0.606%	628.87	27379
gcut12dr	561640000	559820015.8	0.325%	1328.03	32554

Table 8.21: Performance of the algorithm CGVR^p with rotations and 4-staged patterns.

8.6.4 Computational results for the SP problem

For the problem SP, we have used the instances *gcut1d,. . .,gcut12d* considering the maximum distance between two horizontal cuts of the strip as the width of the bin.

Although the instances for the SP problem required considerably more time than the (same) instances for the 2CS problem, the corresponding times required by the latter were still small and acceptable in practice.

The results for algorithm CGS^p for 2-, 3- and 4-staged cutting are shown in tables 8.22, 8.23 and 8.24, respectively. The lower bound corresponds to the optimal fractional solution of formulation 8.2.

Instance	Solution of CGS^p	LB	Difference from LB	Average Time (sec)	Columns Generated	Solution of FFDH	Improvement over FFDH
gcut1d	51604	51583.0	0.041%	0.06	43	54323	5.01%
gcut2d	77436	77369.5	0.086%	0.26	141	77436	0.00%
gcut3d	80206	80112.5	0.117%	4.50	1479	83529	3.98%
gcut4d	196480	196422.5	0.029%	3.74	702	205250	4.27%
gcut5d	91177	91177.0	0.000%	0.04	29	96693	5.70%
gcut6d	168148	167987.5	0.096%	0.18	93	181578	7.40%
gcut7d	243241	243076.0	0.068%	0.65	232	259462	6.25%
gcut8d	332924	332669.3	0.077%	3.57	534	344732	3.43%
gcut9d	122836	122532.5	0.248%	0.08	66	129706	5.30%
gcut10d	272919	272680.5	0.087%	0.22	119	286790	4.84%
gcut11d	315026	314747.5	0.088%	1.50	332	338271	6.87%
gcut12d	573806	573590.0	0.038%	8.88	610	605126	5.18%

Table 8.22: Performance of the algorithm CGS^p with 2-staged patterns.

For the 2-staged problem, all instances were solved in less than 10 seconds. On the average, the difference between the solutions found by the algorithm an the lower bound was only 0.081% and an optimal solution for instance *gcut5d* was found. The improvement of the algorithm CGS^p over FFDH was, on the average, of 4.85%. These improvements are very significant, since algorithm FFDH also produces 2-staged solutions.

For the 3-staged problem, the most difficult instance (*gcut12*) take 151 seconds to be completed. On the average, the difference between the solutions found by the algorithm and the lower bound was 0.113% and the average improvement over FFDH was 7.66%.

For the 4-staged problem, the difference between the solutions found by the algorithm CGS^p an the lower bound was 0.116% and the improvement over FFDH was 7.74%, on the average.

We also performed tests when orthogonal rotations are allowed. The results of the tests can be found in tables 8.25, 8.26 and 8.27. On the average, the difference between the solutions found by the algorithm CGSR^p and the lower bound was 0.114%, 0.204% and 0.261%

Instance	Solution of CGS^p	LB	Difference from LB	Average Time (sec)	Columns Generated	Solution of FFDH	Improvement over FFDH
gcut1d	51432	51332.8	0.193%	0.25	188	54323	5.32%
gcut2d	77436	77369.5	0.086%	0.31	116	77436	0.00%
gcut3d	77790	77728.7	0.079%	10.51	3297	83529	6.87%
gcut4d	195307	195249.5	0.029%	8.40	1233	205250	4.84%
gcut5d	87249	87164.4	0.097%	0.09	61	96693	9.77%
gcut6d	158137	158104.5	0.021%	0.32	132	181578	12.91%
gcut7d	236508	236412.8	0.040%	1.28	319	259462	8.85%
gcut8d	310748	310493.8	0.082%	42.55	4861	344732	9.86%
gcut9d	120479	119988.6	0.409%	0.33	245	129706	7.11%
gcut10d	260388	260259.5	0.049%	0.33	131	286790	9.21%
gcut11d	305348	304918.0	0.141%	9.96	1386	338271	9.73%
gcut12d	559870	559132.5	0.132%	151.08	9748	605126	7.48%

Table 8.23: Performance of the algorithm CGS^p with 3-staged patterns.

respectively for the 2-, 3- and 4-staged problem. Comparing with the solutions generated by the FFDHR2 we obtain on the average an improvement of 11.62%, 13.41% and 13.42% respectively for the 2-, 3- and 4-staged problem.

Instance	Solution of CGS^p	LB	Difference from LB	Average Time (sec)	Columns Generated	Solution of FFDH	Improvement over FFDH
gcut1d	51432	51332.8	0.193%	0.23	178	54323	5.32%
gcut2d	77436	77369.5	0.086%	0.40	126	77436	0.00%
gcut3d	77446	77287.0	0.206%	19.80	4516	83529	7.28%
gcut4d	195307	195249.5	0.029%	9.70	1118	205250	4.84%
gcut5d	87249	87164.4	0.097%	0.11	62	96693	9.77%
gcut6d	158137	158104.5	0.021%	0.40	149	181578	12.91%
gcut7d	236508	236412.8	0.040%	1.48	314	259462	8.85%
gcut8d	310672	310493.8	0.057%	47.28	4544	344732	9.88%
gcut9d	119861	119426.2	0.364%	0.20	131	129706	7.59%
gcut10d	260388	260259.5	0.049%	0.39	132	286790	9.21%
gcut11d	305348	304918.0	0.141%	13.80	1557	338271	9.73%
gcut12d	559159	558531.9	0.112%	201.51	10422	605126	7.60%

Table 8.24: Performance of the algorithm CGS^p with 4-staged patterns.

Instance	Solution of CGSR	LB	Difference from LB	Average Time (sec)	Columns Generated	Solution of FFDHR2	Improvement over FFDHR2
gcut1dr	50612	50589.0	0.045%	0.10	54	54323	6.83%
gcut2dr	60311	60192.0	0.198%	1.18	347	74744	19.31%
gcut3dr	77385	77296.3	0.115%	5.88	1193	83529	7.36%
gcut4dr	175996	175930.4	0.037%	32.11	3501	191383	8.04%
gcut5dr	78530	78370.8	0.203%	0.56	235	96530	18.65%
gcut6dr	138207	138041.0	0.120%	0.97	224	181578	23.89%
gcut7dr	226312	226163.8	0.066%	3.28	531	244742	7.53%
gcut8dr	300696	300499.3	0.065%	29.54	1419	326197	7.82%
gcut9dr	119584	119417.0	0.140%	0.22	102	129657	7.77%
gcut10dr	236531	236278.2	0.107%	1.20	193	265322	10.85%
gcut11dr	286164	285661.6	0.176%	10.53	555	326275	12.29%
gcut12dr	549751	549181.6	0.104%	130.30	2908	605126	9.15%

Table 8.25: Performance of the algorithm CGSR^p with rotations and 2-staged patterns.

Instance	Solution of CGSR	LB	Difference from LB	Average Time (sec)	Columns Generated	Solution of FFDHR2	Improvement over FFDHR2
gcut1dr	50433	50329.0	0.207%	0.44	250	54323	7.16%
gcut2dr	59369	59138.7	0.389%	4.91	1220	74744	20.57%
gcut3dr	75447	75227.5	0.292%	80.03	11692	83529	9.68%
gcut4dr	173796	173588.0	0.120%	127.54	11925	191383	9.19%
gcut5dr	74885	74706.0	0.240%	0.40	140	96530	22.42%
gcut6dr	135952	135450.9	0.370%	4.77	1227	181578	25.13%
gcut7dr	221258	221137.5	0.054%	7.39	847	244742	9.60%
gcut8dr	294465	294188.3	0.094%	353.28	13965	326197	9.73%
gcut9dr	116404	115994.6	0.353%	0.55	231	129657	10.22%
gcut10dr	233321	233253.7	0.029%	2.14	211	265322	12.06%
gcut11dr	278144	277452.3	0.249%	232.55	7009	326275	14.75%
gcut12dr	541926	541610.5	0.058%	1179.61	20669	605126	10.44%

Table 8.26: Performance of the algorithm CGSR^p with rotations and 3-staged patterns.

Instance	Solution of CGSR	LB	Difference from LB	Average Time (sec)	Columns Generated	Solution of FFDHR2	Improvement over FFDHR2
gcut1dr	50433	50329.0	0.207%	0.47	255	54323	7.16%
gcut2dr	59420	59124.5	0.500%	10.75	2222	74744	20.50%
gcut3dr	75396	75162.2	0.311%	50.26	7261	83529	9.74%
gcut4dr	173687	173534.3	0.088%	259.84	20740	191383	9.25%
gcut5dr	74717	74391.0	0.438%	0.46	155	96530	22.60%
gcut6dr	135952	135450.9	0.370%	6.17	1332	181578	25.13%
gcut7dr	221258	221137.5	0.054%	8.19	791	244742	9.60%
gcut8dr	294578	294188.1	0.133%	764.96	23291	326197	9.69%
gcut9dr	116296	115927.8	0.318%	0.51	164	129657	10.30%
gcut10dr	233582	233066.7	0.221%	5.24	376	265322	11.96%
gcut11dr	278362	277230.7	0.408%	206.29	5251	326275	14.68%
gcut12dr	541998	541540.0	0.085%	935.73	12450	605126	10.43%

Table 8.27: Performance of the algorithm CGSR^p with rotations and 4-staged patterns.

8.7 Concluding remarks

In this paper we presented algorithms for the RK, 2CS, BPV and SP problems and their variants RK^r, BP^r, BPV^r and SP^r (where orthogonal rotations of the items are allowed) using guillotine staged patterns.

For the RK problem we presented the (exact) pseudo-polynomial algorithms SDP for k-staged patterns. We have also mentioned how to use SDP to solve the problem RK^r.

We extend the work of [18] by using column generation based algorithms to solve the 2CS and BPV problems using staged patterns, and also extended these algorithms to solve the SP problem. These algorithms use, as subroutines, the algorithm SDP to generate the columns. The algorithms combines different techniques: Simplex method with column generation, an exact algorithm for the discretization points, and approximation algorithms for the last residual instance. An approach of this nature has shown to be promising, and has been used to tackle the one-dimensional cutting stock problem [48, 17].

The algorithm for the SP problem was obtained adapting the algorithm for the BPV problem. We have used the same strategy used in the algorithms for the 2CS and BPV problems. The residual instances were solved with an approximation algorithm (FFDH) or another algorithm we proposed (called FFDHR2) when rotations are allowed.

For almost all instances tested, the algorithms that use a perturbation method found solutions of a slightly better quality than CG (respectively CGR) at the cost of a slight increase in the running time.

A natural development of our work would be to adapt the approach used in the algorithm CG for the version with arbitrary orthogonal cutting patterns (the cuts need not be guillotine). One can find an initial solution using homogeneous patterns; the columns can be generated using any of the algorithms that have appeared in the literature for the two-dimensional cutting stock problem with value [6, 3]. To solve the last residual instance one can use approximation algorithms [16, 11, 34].

One can also use column generation for the variant of 2CS in which the quantity of items in each bin is bounded. This variant, proposed by Christofides and Whitlock [15], is called *restricted two-dimensional cutting stock problem*. Each new column can be generated with any of the known algorithms for the restricted two-dimensional cutting stock problem with value [15, 41], and the last residual instance can be solved with the algorithm M-HFF. This restricted version with guillotine cut requirement can also be solved using the ideas we have just described: the homogeneous patterns and the patterns produced by M-HFF can be obtained with guillotine cuts, and the columns can be generated with the algorithm of Cung, Hifi and Le Cun [24].

As a final remark we mention that we did not use a heuristic procedure to solve the column generation step. Therefore, we could obtain optimal fractional solutions for all the instances

we have considered. These optimal fractional solutions yielded excellent lower bounds for the optimal solutions, which turned out to be in most of the tests, very close to the solutions found by the algorithms.

We performed many tests and compared the solutions obtained for the different variants of the problems. On average, we noted an increase in computational time and decrease of space occupation when we considered 2-, 3- and 4-staged patterns, as well as when rotations were considered. It is interesting to note that very few papers consider 4-staged patterns. Finally, we observe that for all tests performed, the algorithms we implemented found optimal or quasi-optimal solutions in a reasonable amount of time, showing that they may be useful for practical purposes

8.8 Bibliography

1. Ilan Adler, Nimrod Megiddo, and Michael J. Todd. New results on the average behavior of simplex algorithms. *Bull. Amer. Math. Soc. (N.S.)*, 11(2):378–382, 1984.

2. Ramon Alvarez-Valdes, Antonio Parajon, and Jose M. Tamarit. A computational study of LP-based heuristic algorithms for two-dimensional guillotine cutting stock problems. *OR Spektrum*, 24(2):179–192, 2002.

3. M. Arenales and R. Morábito. An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems. *European Journal of Operational Research*, 84:599–617, 1995.

4. B. S. Baker, D. J. Brown, and H. P. Katseff. A $\frac{5}{4}$ algorithm for two-dimensional packing. *Journal of Algorithms*, 2:348–368, 1981.

5. J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36(4):297–306, 1985.

6. J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.

7. J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

8. G. Belov and G. Scheithauer. Models with variable strip widths for two-dimensional two-stage cutting. www.math.tu-dresden.de/ ~capad/PAPERS/03-varwidth.pdf, 2003.

9. Karl-Heinz Borgwardt. Probabilistic analysis of the simplex method. In *Mathematical developments arising from linear programming (Brunswick, ME, 1988)*, volume 114 of *Contemp. Math.*, pages 21–34. Amer. Math. Soc., Providence, RI, 1990.

10. Andreas Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172:814–837, 2006.

11. A. Caprara. Packing 2-dimensional bins in harmony. In *Proc. of 43rd Symposium on Foundations of Computer Science*, pages 490–499, 2002.

12. A. Caprara, A. Lodi, and M Monaci. An approximation scheme for the two-stage, two-dimensional bin packing problem. In A.S. Schulz W.J. Cook, editor, *Proceedings of the Ninth Conference on Integer Programming and Combinatorial Optimization (IPCO'02)*, pages 320–334. Springer-Verlag, 2002.

13. A. Caprara, A. Lodi, and M. Monaci. Fast approximation schemes for two-stage, two-dimensional bin packing. *Mathematics of Operations Research*, 30(1):150–172, 2005.

14. A. Caprara and M. Monaci. On the two-dimensional knapsack problem. *Operations Research Letters*, 32:5–14, 2004.

15. N. Christofides and C. Whitlock. An algorithm for two dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

16. F. R. K. Chung, M. R. Garey, and D. S. Johnson. On packing two-dimensional bins. *SIAM J. Algebraic and Discrete Methods*, 3:66–76, 1982.

17. G. F. Cintra. Algoritmos híbridos para o problema de corte unidimensional. In *XXV Conferência Latinoamericana de Informática*, Assunção, 1999.

18. G. F. Cintra. *Algoritmos para problemas de corte de guilhotina bidimensional*. PhD thesis, Instituto de Matemática e Estatística, São Paulo, 2004.

19. G. F. Cintra, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier. Approximation algorithms for cutting stock problems. *European Journal of Operational Research*, to appear.

20. G. F. Cintra and Y. Wakabayashi. Dynamic programming and column generation based approaches for two-dimensional guillotine cutting problems. In *Proceedings of WEA 2004: Workshop on Efficient and Experimental Algorithms*, volume 3059 of *Lecture Notes in Computer Science*, pages 175–190. 2004.

21. 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*, 9:808–826, 1980.

22. COIN-OR Linear Program Solver. An Open Source code for solving linear programming problems, http://www.coin-or.org/Clp/index.html.

23. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

24. 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(3):185–210, 2000.

25. L. Epstein. Two dimensional packing: The power of rotation. In *Proc. of the 28th International Symposium of Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes on Computer Science – LNCS*, pages 398–407. Springer–Verlag, 2003.

26. S. P. Fekete and J. Schepers. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, to appear.

27. P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

28. P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem - part II. *Operations Research*, 11:863–888, 1963.

29. P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

30. M. Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, 25(11):925–940, 1998.

31. J. C. Herz. A recursive computational procedure for two-dimensional stock-cutting. *IBM J. Res. Dev.*, pages 462–469, 1972.

32. K. Jansen and R. van Stee. On strip packing with rotations. In *Proc. of the ACM Symposium on Theory of Computing*, pages 755–761, 2005.

33. K. Jansen and G. Zhang. On rectangle packing: maximizing benefits. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 204–213, 2004.

34. C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.

35. A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8:363–379, 2004.

36. A. Lodi and M. Monaci. Integer linear programming models for 2-staged two-dimensional knapsack problem. *Mathematical Programming*, 94:257–278, 2003.

37. S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

38. F. K. Miyazawa and Y. Wakabayashi. Packing problems with orthogonal rotations. In *Proc. of Latin American Theoretical INformatics.*, volume 2976 of *Lecture Notes in Computer Science*, pages 359–368, Buenos Aires, Argentina, 2004. Springer-Verlag.

39. R. Morábito, M. Arenales, and V. F. Arcaro. An and-or-graph approach for two-dimensional cutting problems. *European Journal of Operational Research*, 58:263–271, 1992.

40. E. A. Mukhacheva and A. S. Mukhacheva. The rectangular packing problem: Local optimum search methods based on block structures. *Automation and Remote Control*, 65(2):101–112, 2004.

41. J. F. Oliveira and J. S. Ferreira. An improved version of Wang's algorithm for two-dimensional cutting problems. *European Journal of Operational Research*, 44:256–266, 1990.

42. J. Puchinger and G. R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2d bin packing. In *Proc. of Parallel Problem Solving from Nature - PPSN VIII*, volume LNCS 3242, pages 642–651. Springer-Verlag, 2004.

43. J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, to appear.

44. J. R. Riehme, G. Scheithauer, and J. Terno. The solution of two-stage guillotine cutting stock problems having extremely varying order demands. *European Journal of Operational Research*, 91:543–552, 1996.

45. S. S. Seiden and G. J. Woeginger. The two-dimensional cutting stock problem revisited. *Mathematical Programming*, 102(3):519–530, 2005.

46. F. Vanderbeck. A nested decomposition approach to a 3-stage 2-dimensional cutting stock problem. *Management Science*, 47:864–879, 2001.

47. P. Y. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31:573–586, 1983.

48. G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: a computational study. *OR Spektrum*, 18:131–144, 1996.

49. G. Wäscher, H. Haussner, and H. Schumann, An improved typology of cutting and packing problems, *European Journal of Operational Research*, to appear.

Capítulo 9

Conclusões e Trabalhos Futuros

Neste trabalho apresentamos algoritmos para diversos problemas de empacotamento. O principal foco do trabalho foi o desenvolvimento de algoritmos de aproximação e heurísticas baseadas no método de geração de colunas.

No Capítulo 4 apresentamos o problema que chamamos de *Class Constrained Shelf Bin Packing* (CCSBP). Este problema é uma generalização do problema *bin packing* onde itens têm classes diferentes e devemos empacotar os itens separando-os por prateleiras. Este problema possui aplicações na indústria de metais [16]. Apresentamos algoritmos aproximados *práticos* para este problema, e também um esquema de aproximação para o caso em que o número de classes diferentes é limitado por uma constante. Como trabalho futuro permanece em aberto a questão da existência de um esquema de aproximação para o caso onde o número de classes faz parte da entrada.

No Capítulo 5 consideramos dois problemas: o CCSBP e o problema *bin packing* com restrições de classes. Apresentamos esquemas de aproximação duais para ambos os problemas. Neste caso buscamos soluções para a versão dual, que podem ser inviáveis para o problema original, usando no máximo a quantidade de recipientes de uma solução ótima da versão original. A medida da qualidade da solução gerada está relacionada com o grau de inviabilidade da solução. Como possível trabalho futuro pode-se tentar propor esquemas de aproximações duais com complexidade de tempo mais baixa, já que a complexidade de tempo dos algoritmos propostos é muito alta.

No Capítulo 6 apresentamos o problema *bin packing* com restrição de classes, denotado por CCBP, com aplicações para um problema de construção de servidores de vídeo sob demanda. Apresentamos algoritmos aproximados práticos para este problema e exibimos resultados de testes computacionais com tais algoritmos. Também apresentamos algoritmos aproximados para a versão *online* do problema. Por fim, apresentamos um esquema de aproximação para o caso em que o número de classes diferentes da entrada é limitado por uma constante. Aqui também fica em aberto a existência de um esquema de aproximação quando o número de classes

diferentes faz parte da entrada. Um esquema de aproximação para este problema pode levar a um esquema de aproximação para o problema CCSBP. Por outro lado, um resultado de inaproximabilidade também pode ajudar a construir um resultado semelhante para o problema CCSBP.

No Capítulo 7 apresentamos algoritmos de aproximação para a versão do problema *bin packing* onde os itens possuem demandas, problemas que são conhecidos na literatura como problemas de *cutting stock*. Neste capítulo mostramos como adaptar vários algoritmos de aproximação desenvolvidos para problemas sem demanda para o caso onde há demanda para os itens. Dentre os resultados deste capítulo destacamos um esquema de aproximação assintótico para o problema *cutting stock* unidimensional e um algoritmo com fator de aproximação assintótico 2.077 para o problema *cutting stock* bidimensional. Neste ponto, destacamos que a complexidade computacional do problema de *cutting stock* está em aberto. Apesar de sabermos que este problema é NP-difícil, não se sabe se a versão de decisão do problema está em NP.

Finalmente no Capítulo 8, apresentamos algoritmos para problemas de empacotamento bidimensional. Nestes problemas são considerados cortes guilhotináveis e em estágios. Apresentamos algoritmos exatos para problema da mochila bidimensional baseados em programação dinâmica. Consideramos também o problema *bin packing* bidimensional com demandas e o problema *strip packing* bidimensional com demandas. Para estes problemas apresentamos heurísticas baseadas no método de geração de colunas. Um possível ponto para trabalhos futuros é estender os algoritmos descritos para problemas tridimensionais. Para tanto, deve-se construir algoritmos eficientes para o problema da mochila tridimensional, que será usada na parte de geração de colunas.

Algoritmos de aproximação são vistos por muitas pessoas como resultados teóricos sem grande aplicabilidade prática. Nesta tese apresentamos alguns algoritmos aproximados práticos e fizemos alguns testes computacionais (ver Capítulo 6). Os resultados destes testes mostram que tais algoritmos produzem soluções de excelente qualidade, podendo ser utilizados na prática. Vários algoritmos aproximados são de fácil implementação e em geral produzem soluções cujos valores estão muito mais próximos do ótimo do que os fatores de aproximação demonstrados [39, 43]. Tais algoritmos podem ser usados inclusive como heurísticas primais em algoritmos exatos (veja [1] como exemplo).

Também há um grande interesse de investigação teórica relacionada a algoritmos de aproximação. Neste caso busca-se saber, para um determinado problema, qual o melhor fator de aproximação que pode ser obtido por um algoritmo para este problema. Nesta linha pode-se projetar algoritmos aproximados ou provar resultados de inaproximabilidade [25, 5, 39]. A partir de tais resultados criou-se uma teoria de complexidade baseada em classes de aproximação (veja por exemplo [5]). Tais resultados trazem uma maior fundamentação teórica para a questão de se P é igual a NP. Hoje sabemos, por exemplo, que a existência de algum FPTAS para uma enorme quantidade de problemas equivaleria a mostrar que $\mathrm{P} = \mathrm{NP}$. Com os recentes resul-

tados sobre provas verificáveis probabilisticamente [2, 3, 4], sabemos que diversos problemas não admitem sequer aproximação constante a menos que $\mathrm{P} = \mathrm{NP}$.

Nesta tese também investigamos heurísticas baseadas no método de geração de colunas. A grande vantagem desta abordagem é trabalhar com programas lineares que fornecem limitantes duais muito próximos do ótimo. Para o problema *cutting stock* unidimensional existe uma conjectura famosa, para a formulação correspondente a apresentada no Capítulo 7 (formulação 8.1), conhecida como MIRUP (*Modified Integer Round-Up Property*) que diz o seguinte: O valor de uma solução inteira ótima para uma instância I do *cutting stock* unidimensional é no máximo o teto da função objetivo do programa linear adicionado de 1. Vários resultados corroboram com esta conjectura [34, 35, 40]. Para o caso bidimensional com cortes em dois estágios Riehme *et al.* [33] apresentam uma versão da conjectura MIRUP, (mas neste caso adiciona-se 2 ao valor do teto do programa linear) e resultados computacionais dando suporte a conjectura. Uma maior investigação sobre a qualidade do limitante dual fornecido pela formulação 8.1 para o problema bidimensional pode ser um interessante trabalho. Como vimos no Capítulo 7, as heurísticas propostas, baseadas na solução deste programa linear, obtiveram excelentes resultados.

Referências Bibliográficas

[1] C. E. Andrade, F. K. Miyazawa, and E. C. Xavier. Um algoritmo exato para o problema de empacotamento bidimensional em faixa. In *Anais do XXXVIII Simpósio Brasileiro de Pesquisa Operacional*, volume em CD, pages 1–12, 2006.

[2] S. Arora and C. Lund. *Hardness of approximations*. PSW Publishing, 1997.

[3] S. Arora, C. Lund, R. Motwani, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33th IEEE Annual Symposium on Foundations of Computer Sciense*, pages 106–113, 1998.

[4] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. In *Proc. 33th IEEE Annual Symposium on Foundations of Computer Sciense*, pages 2–13, 1992.

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation: combinatorial optimization problems and their approximality properties*. Springer-Verlag, 1999.

[6] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 1990.

[7] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.

[8] G. F. Cintra, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier. A note on the approximability of cutting stock problems. European Journal of Operational Research, to appear, http://dx.doi.org/10.1016/j.ejor.2005.09.053.

[9] G. F. Cintra and Y. Wakabayashi. Dynamic programming and column generation based approaches for two-dimensional guillotine cutting problems. In *Proceedings of WEA 2004: Workshop on Efficient and Experimental Algorithms*, volume 3059 of *Lecture Notes in Computer Science*, pages 175–190. 2004.

[10] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 2, pages 46–93. PWS, 1997.

[11] M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. Technical report, IBM, T.J. Watson Research Center, NY, 1998.

[12] M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. In *Proceedings of the 1th Brazilian Symposium on Graph Algorithms and Combinatorics*, volume 7 of *Electronic Notes in Discrete Mathematics*, 2001.

[13] J. P. dos Santos and M. P. Mello. *Introdução a Análise Combinatória*. Unicamp, 2002.

[14] H. Dyckhoff. A typology of cutting and packing problems. *European J. Operational Research*, 44:145–159, 1990.

[15] C. E. Ferreira, C. G. Fernandes, F. K. Miyazawa, J. A. R. Soares, J. C. Pina Jr., K. S. Guimarães, M. H. Carvalho, M. R. Cerioli, P. Feofiloff, R. Dahab, and Y. Wakabayashi. *Uma introdução sucinta a algoritmos de aproximação*. Colóquio Brasileiro de Matemática – IMPA, Rio de Janeiro–RJ, 2001.

[16] J. S. Ferreira, M. A. Neves, and P. Fonseca e Castro. A two-phase roll cutting problem. *European J. Operational Research*, 44:185–196, 1990.

[17] S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continous media servers. *Parallel Computing Journal*, 24(1):91–122, 1998.

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

[19] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem - part II. *Operations Research*, 11:863–888, 1963.

[20] P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

[21] P. Gilmore and R. Gomory. The theory and computation of knapsack functions. *Operations Research*, 15:1045–1075, 1967.

[22] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42:1115–1145, 1995.

[23] T. C. Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162(3):1065–1185, 2005.

[24] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *journal of the ACM*, 34(1):144–162, 1987.

[25] D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.

[26] O.H. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the Association for Computing Machinery*, 22:463–468, 1975.

[27] C. Kenyon and E. Rémila. Approximate strip packing. In *37th Annual Symposium on Foundations of Computer Science*, pages 31–36, 1996.

[28] C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.

[29] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete and Applied Mathematics*, 123:379–396, 2002.

[30] A. Lodi, S. Martello, and D. Vigo. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[31] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc, 1990.

[32] Silvano Martello, Michele Monaci, and Daniele Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[33] J. R. Riehme, G. Scheithauer, and J. Terno. Numerical investigations on the mirup of the 2-stage guillotine cutting stock problem. Technical Report MATH-NM-17-1995, Techn. Univ. Dresden, 1995.

[34] G. Scheithauer and J. Terno. The modified integer round-up property of the one-dimesional cutting stock problem. *European J. Operational Research*, 84:562–571, 1995.

[35] G. Scheithauer and J. Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Oper. Res. Lett.*, 20:93–100, 1997.

[36] H. Shachnai and T. Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, 32(4):651–678, 2002.

[37] H. Shachnai and T. Tamir. Tight bounds for online class-constrained packing. *Theoretical Computer Science*, 321(1):103–123, 2004.

[38] N. J. A. Sloane. Kepler's conjecture confirmed. *Nature*, 395(6701):435, 1998.

[39] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[40] G. Wäscher and T. Gau. Two approaches to the cutting stock problem. In *IFORS 93 Conference*, 1993.

[41] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. To Appear in European Journal of Operational Research.

[42] J. L. Wolf, P. S. Wu, and H. Shachnai. Disk load balancing for video-on-demand-systems. *ACM Multimedia Systems Journal*, 5:358–370, 1997.

[43] E. C. Xavier and F. K. Miyazawa. Practical comparison of approximation algorithms for scheduling problems. *Pesquisa Operacional*, 24(2):227–252, 2004.

[44] E. C. Xavier and F. K. Miyazawa. A one-dimensional bin packing problem with shelf divisions. In *2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics (GRACO 2005)*, volume 19 of *Electronic Notes in Discrete Mathematics*, 2005.

[45] E. C. Xavier and F. K. Miyazawa. The class constrained bin packing problem with applications to video-on-demand. In *Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON'06)*, volume 4112 of *Lecture Notes in Computer Science*, pages 439–448, 2006.