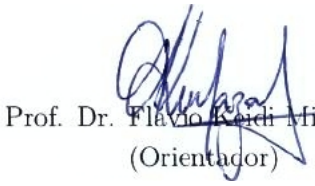
Problema de Empacotamento em Faixa com Restrições de Ordem e Estabilidade

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fabrício Luis Santos da Silva e aprovada pela Banca Examinadora.

Campinas, 17 de Dezembro de 2010.



Prof. Dr. Flávio Keidi Miyazawa
(Orientador)

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

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Silva, Fabrício Luís Santos da
Si38p Problema de empacotamento em faixa com restrições de ordem e estabilidade/Fabrício Luís Santos da Silva-- Campinas, [S.P. : s.n.], 2010.

Orientador : Flávio Keidi Miyazawa.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

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

Título em inglês: Strip packing problem with constraints in order and stability

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

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Flávio Keidi Miyazawa (IC – UNICAMP)
Prof. Dr. Eduardo Candido Xavier (IC – UNICAMP)
Prof. Dr. Reinaldo Morabito Neto (PPGEP - UFSCar)

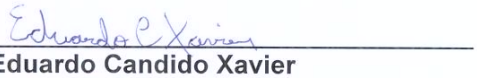
Data da defesa: 17/12/2010

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 17 de dezembro de 2010, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Reinaldo Morabito Neto
Centro de Ciência Exatas e Tecnológica / UFScar



Prof. Dr. Eduardo Candido Xavier
IC / UNICAMP

Prof. Dr. Flávio Keidi Miyazawa
IC / UNICAMP

III

Instituto de Computação
Universidade Estadual de Campinas

Problema de Empacotamento em Faixa com Restrições de Ordem e Estabilidade

Fabrício Luis Santos da Silva

Dezembro de 2010

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa (Orientador)
- Prof. Dr. Eduardo Candido Xavier
  Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)
- Prof. Dr. Reinaldo Morabito Neto
  Departamento de Engenharia de Produção, Universidade Federal de São Carlos (UFSCar)
- Prof. Dr. Luis Augusto Angelotti Meira (suplente)
  Departamento de Ciência e Tecnologia, Universidade Federal de São Paulo (UNIFESP)
- Prof. Dr. Orlando Lee (suplente)
  Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)

Resumo

Neste trabalho lidamos com o problema de Empacotamento em Faixa Bidimensional considerando o caso em que os itens devem ser dispostos de forma a manter o empacotamento estável e satisfazer uma ordem de descarregamento imposta. Consideramos o caso em que a orientação dos itens é fixa. Definimos uma metodologia para analisar a estabilidade do empacotamento observando as condições de equilíbrio estático para corpos rígidos. Desenvolvemos heurísticas e formulamos um programa linear inteiro para o problema de Empacotamento em Faixa sujeito a tais restrições. A resolução da formulação inteira ocorre através de uma estratégia do tipo *branch-and-cut*. As restrições de estabilidade foram inseridas como planos de corte de maneira a remover empacotamentos que não são estáveis. Em nossos experimentos computacionais, vemos que o modelo proposto é adequado para lidar com instâncias de pequeno até médio porte, dentro de um tempo computacional razoável.

Palavras-Chave: Problema de Empacotamento em Faixa Bidimensional. Programa Linear Inteiro. Equilíbrio Estático de Corpos Rígidos.

Abstract

This paper investigates the Two-Dimensional Strip Packing Problem considering the case in which the items should be arranged to form a stable packing and satisfy an order of unloading, so that after unloading, the packing is still stable. We consider the case where the items are oriented and rotations are not allowed. We present a methodology to analyze the stability of the packing observing the conditions for static equilibrium of rigid bodies. We present heuristics and formulate an integer linear programming model for the Strip Packing problem considering such constraints. To solve the integer formulation, we develop a branch-and-cut approach. For each integer solution obtained during the branch-and-cut algorithm, corresponding to a non-stable packing, we insert a cutting plane for which this integer solution is not satisfied. In our computational experiments, we see that the proposed model is suitable to deal with small and mid-sized instances. Some optimal solutions were obtained after few hours of CPU processing.

Keywords: Two-Dimensional Strip Packing Problem. Integer Linear Programming. Static Equilibrium of Rigid Bodies.

Agradecimentos

A minha família pela motivação, compreensão e força, com que sempre me acompanharam.

Ao meu orientador Professor Dr. Flávio Keidi Miyazawa por causa da sua dedicação, disponibilidade, atenção, paciência e excelente orientação.

A todos meus amigos pelo apoio, carinho e inúmeros momentos de felicidade.

Aos meus amigos do IC pelos grandes momentos de estudo e reflexão.

Aos professores do IC e ao pessoal do CPG pelo apoio fornecido durante o mestrado.

Aos professores do Departamento de Física e Engenharia Civil pela disponibilidade e ajuda com as dúvidas sobre a Estática.

A Unicamp pela sua ótima infraestrutura.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico pelo apoio financeiro prestado.

E a todas as pessoas do IC que me proporcionaram uma experiência única e gratificante de mestrado.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O problema de empacotamento em faixa consiste em empacotar vários objetos retangulares menores dentro de um objeto retangular maior de altura ilimitada com o objetivo de minimizar a altura do empacotamento formado. Neste trabalho, os objetos retangulares menores são denominados de itens e o objeto retangular maior de recipiente. Os itens serão empacotados ortogonalmente e possuirão orientação fixa dentro do empacotamento.

O problema de empacotar itens tem várias aplicações práticas na indústria. Uma restrição real deste problema envolve os itens serem organizados de forma *estável* dentro do recipiente, isto é, os itens devem satisfazer as condições de equilíbrio estático dos corpos rígidos, tal que não podem cair ou desmanchar o empacotamento depois de feito. Outra restrição envolve a *ordem* como os itens devem ser empacotados dentro do recipiente. A ordem ou prioridade de descarregamento garante que um item será empacotado numa posição que respeita a prioridade dos outros itens do empacotamento.

O objetivo deste trabalho é resolver o problema de empacotamento em faixa bidimensional considerando as restrições de ordem e a estabilidade estática.

No desenvolvimento deste projeto utilizamos conceitos físicos aplicados na Física e na Engenharia Civil para verificar a estabilidade de um empacotamento. Elaboramos uma formulação linear que resolve o problema de empacotamento em faixa com restrição de ordem e que favorece encontrar empacotamentos estáveis. Para cada instância do problema, criamos um programa linear que será resolvido utilizando um algoritmo *branch-and-cut* que usa como plano de corte desigualdades geradas pelo algoritmo que verifica a estabilidade. Além disso, desenvolvemos heurísticas que geram empacotamentos estáveis.

Este trabalho está estruturado em seis capítulos.

O capítulo 2 mostra mais detalhes do problema realizando uma revisão literária sobre problemas relacionados a empacotamentos. Os problemas de empacotamentos serão di-

vididos em duas variantes. Na primeira variante, encontram-se os empacotamentos para os quais a ordem não é importante e na segunda temos os problemas para os quais ela importa.

O capítulo 3 descreve os conceitos físicos e como procede a verificação da estabilidade. Esta última se dá verificando individualmente cada item do empacotamento, analisando as forças que atuam neste item e as forças que devem ser repassadas para os itens que estão em contato direto.

O capítulo 4 traz a modelagem do problema em um programa linear inteiro. Este capítulo apresenta detalhes do domínio deste problema, das formulações desenvolvidas e do algoritmo de empacotamento.

O capítulo 5 mostra as heurísticas que foram desenvolvidas e utilizadas. Já no capítulo 6 encontra-se os resultados e análise dos testes. E por fim, o capítulo 7 que contém as conclusões deste trabalho.

Capítulo 2

Problema

Neste capítulo apresentamos o problema de empacotamento de itens. Inicialmente, daremos uma pequena introdução sobre o que são problemas de empacotamento. Posteriormente faremos uma breve revisão literária e mostraremos em detalhes o problema de empacotamento em faixa bidimensional.

2.1 Introdução

O problema de empacotamento de itens ocorre quando desejamos colocar itens dentro de um ou vários recipientes respeitando um determinado objetivo. Recipiente é o local onde os itens serão empacotados ou armazenados e os itens são objetos que devem ser empacotados dentro do recipiente. Neste problema tanto os itens como os recipientes são retangulares e podem ser de vários tamanhos.

O objetivo do problema depende do número de recipientes e de itens que serão analisados. Quando temos um recipiente e vários itens podemos ter como objetivo a minimização da altura do empacotamento ou maximização da área ou do volume do empacotamento resultante. Além dos objetivos citados anteriormente para um recipiente, quando possuímos vários recipientes e vários itens podemos ter outros objetivos como minimizar o número de recipientes utilizados para empacotar todos os itens ou minimizar a soma das áreas ou dos espaços que sobram em cada recipiente.

Os problemas de empacotamento podem ser unidimensionais, bidimensionais ou tridimensionais. Na literatura encontramos o problema da mochila [30] que equivale a preencher um recipiente com itens de diferentes valores e pesos com o objetivo de preencher o recipiente com o maior conjunto de valores possível de modo que este recipiente não ultra-

passe um peso máximo determinado. Também encontramos o empacotamento bidimensional que equivale a empacotar itens bidimensionais dentro de um recipiente bidimensional satisfazendo a um objetivo, como por exemplo, minimizar a altura do empacotamento. E o empacotamento tridimensional que equivale a encontrar as posições em que itens tridimensionais devem ser colocados dentro de um recipiente tridimensional respeitando um objetivo de minimização ou maximização compatível com o problema. De modo geral, muito destes problemas são $\mathcal{NP}$-Difíceis e são difíceis de serem resolvidos na prática.

Os itens a serem empacotados podem variar pela sua forma e quanto a sua variedade. Quanto à forma, elas podem ser regulares ou irregulares. Como exemplos de formas regulares temos os retângulos, caixas, círculos, esferas e etc. Já as formas irregulares são normalmente assimétricas. Quanto à variedade dos itens, podemos tentar empacotar itens de comprimentos iguais ou itens de comprimentos diferentes. Os itens devem ser empacotados ortogonalmente, isto é, cada lado de um item empacotado estará paralelo a cada um dos lados do recipiente.

Buscando padronizar os possíveis tipos de empacotamento e suas variações, Wäscher *et al.* [41] criou uma tipologia que nomeia e enquadra cada problema a uma situação e nomenclatura específica.

2.2 Revisão Literária

Nesta seção apresentamos uma breve revisão literária. Uma das primeiras tentativas de modelar o problema de empacotamento bidimensional foi feita por Gilmore & Gomory [22]. Eles utilizaram uma abordagem de geração de colunas usando uma formulação que considera todos os subconjuntos de itens que podem ser empacotados dentro de um recipiente.

Beasley [2] apresentou um modelo de programação inteira para o problema de corte bidimensional no qual temos um retângulo grande que deve ser cortado em vários itens retangulares de diferentes dimensões com o objetivo de maximizar o valor total dos itens retangulares que foram cortados.

Os problemas de corte e empacotamento são semelhantes, embora na prática estes problemas sejam distintos por causa das diferentes restrições práticas imposta para cada tipo de problema. Do ponto de vista matemático não importa se o padrão obtido para um dado conjunto de unidades é interpretado como sendo um padrão de corte ou um padrão de empacotamento, o que implica na existência de uma *dualidade* entre os problemas de corte e os problemas de empacotamento, isto é, na dualidade entre cortar material/empacotar

espaço e cortar espaço/empacotar material.

Coffman *et al.* [14] tratam o problema de empacotamento em faixa que dado um recipiente e vários itens deve-se empacotar todos os itens neste recipiente de modo a minimizar a altura do empacotamento obtido. Em Lodi *et al.* [32] é apresentado uma resenha para vários tipos de problemas de empacotamento bidimensional.

Kenyon & Rémila [31] apresentaram um AFPTAS (Esquema de Aproximação Assintótico Completamente Polinomial) para o caso orientado de um problema de empacotamento em faixa e Jansen & van Stee [28] propuseram um PTAS (Esquema de Aproximação em Tempo Polinomial) para o caso em que os itens podem ser rotacionados. Os autores Hifi [26] e Lodi *et al.* [33] apresentaram algoritmos do tipo *branch-and-bound* e programas lineares inteiros para problema de empacotamento bidimensionais. Cintra *et al.* [12] propuseram um algoritmo baseado em geração de colunas para o caso com estágios de corte considerando as versões sem e com rotações dos itens para problemas de empacotamento bidimensionais.

Em 1992, Corcoran & Wainwright [15] apresentaram um algoritmo baseado em uma solução genética que tinha como a idéia principal resolver o problema do empacotamento tridimensional reduzindo o para um problema de faixas bidimensionais. Em 1999, Pisinger *et al.* [37] mostraram uma abordagem que ficou conhecida como uma busca local guiada para resolver problemas de otimização combinatória que foi adaptada para resolver problemas de empacotamento tridimensional.

Martello *et al.* [34] apresentaram um algoritmo exato para o problema do empacotamento tridimensional utilizando o método branch-and-bound e duas heurísticas. Autores como Chen *et al.* [9], Christofides & Hadjiconstantinou [10] e Beasley [4] também apresentaram formulações matemáticas que resolvem problemas de empacotamento tridimensionais de forma exata.

Para a versão do problema de empacotamento em faixa tridimensional, temos um algoritmo de aproximação com fator assintótico de $2+\epsilon$ proposto por Jansen & Solis-Oba [27]. Posteriormente, este fator foi melhorado para 1,691 por Bansal *et al.* [1].

Segundo Dowsland *et al.* [17], os problemas de empacotamento são difíceis de serem resolvidos na prática. Esta dificuldade esta relacionada à complexidade destes problemas. Desta forma, não são esperados algoritmos de tempo polinomial para resolvê-los [18].

Grande parte dos trabalhos encontrados na literatura tratam e buscam resolver apenas os problemas inerentes ao empacotamento como o fato de não haver sobreposição entre os itens empacotados e empacotar os itens dentro de um recipiente. Desta forma as soluções encontradas nestes trabalhos garantem que os itens empacotados estarão dentro de algum

recipiente e que não haverá sobreposição entre os itens empacotados.

Trabalhos como os de Castro Silva *et al.* [16] e Junqueira *et al.* [29] além de tratar as condições inerentes ao empacotamento, também consideram a estabilidade.

2.3 Problema de Empacotamento em Faixa Bidimensional

Neste trabalho será abordado o problema de empacotamento em faixa bidimensional considerando a restrição de ordem e a estabilidade estática do empacotamento. Na literatura, desprezando as considerações de restrição de ordem e estabilidade, este problema é conhecido como *Strip Packing Problem*. Já na tipologia de Wäscher *et al.* [41], ele é referenciado como *Two-dimensional Rectangular Open Dimension Problem*.

O problema de empacotamento em faixa bidimensional, que denotaremos por 2EF, é computacionalmente difícil, e consiste em empacotar os itens dentro de um único recipiente de largura fixa e altura infinita buscando minimizar a altura do empacotamento final. Nesta dissertação consideramos que os itens possuem formato retangular, orientação fixa, devem ser empacotados ortogonalmente e todo empacotamento obtido deverá respeitar as condições de equilíbrio dos corpos rígidos.

As condições de estabilidade requerem que o empacotamento obtido respeite as condições de equilíbrio estático dos corpos rígidos [5, 38, 39] garantindo que o empacotamento obtido não se desmanchará dentro do recipiente. Para gerar empacotamentos estáveis Castro Silva *et al.* [16] criaram uma heurística que verifica a estabilidade de um dado empacotamento. Já Junqueira *et al.* [29] tratam a estabilidade através de uma formulação linear inteira.

A heurística de Castro Silva *et al.* [16] funciona da seguinte forma: Primeiro ordena-se todos os itens em ordem decrescente de volume. Em seguida, testa um item de cada vez verificando se é possível inseri-lo dentro do recipiente de modo a obter um empacotamento que seja estaticamente estável. Se o resultado deste teste for positivo, este item é inserido dentro do recipiente. Este processo é repetido para todos os itens restantes. Caso contrário, este item será inserido numa outra oportunidade dentro do empacotamento, pois no momento sua inserção provoca a instabilidade do empacotamento. Após todos os itens terem sido empacotados, um empacotamento estável é obtido.

Junqueira *et al.* [29] tratam a estabilidade através de uma formulação linear inteira criando um parâmetro de estabilidade que amarra a área de contato entre a face infe-

rior de um item e a face superior de um ou mais itens do empacotamento que estejam imediatamente abaixo do item em análise.

A estabilidade neste trabalho é tratada de maneira distinta das encontradas na literatura. Verificamos a estabilidade do empacotamento estudando o equilíbrio individual de cada item no empacotamento. Castro Silva *et al.* [16] analisam a estabilidade do empacotamento olhando todos os itens de uma única vez. Já Junqueira *et al.* [29], consideram a estabilidade do empacotamento através da relação existente entre um item do empacotamento e os itens que lhe dão suporte. O capítulo 3 mostra detalhes sobre a estabilidade.

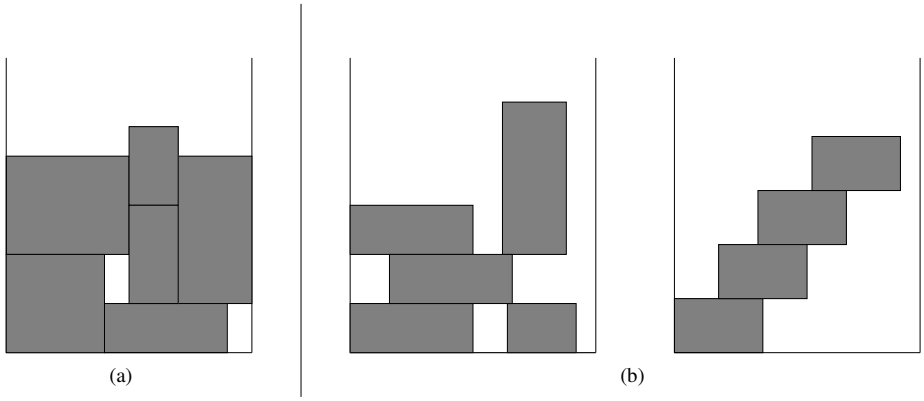


Figura 2.1: Exemplos de empacotamentos: (a) estável e (b) instáveis.

A figura 2.1.(a) mostra um exemplo de empacotamento estável, enquanto a figura 2.1.(b) mostra exemplos de empacotamentos instáveis.

Sucintamente, para verificar a estabilidade o algoritmo considera que cada item possui o seu próprio peso, identifica as forças que poderão vir de itens que estão imediatamente acima e ao juntar o peso do próprio item com as possíveis forças oriundas dos itens que estão acima ele descobre o centro de massa deste item. Com o centro de massa identificado, basta verificar se este centro encontra-se numa região viável. Caso o centro de massa localize-se nesta região, este item satisfaz a estabilidade estática. Em seguida, o algoritmo calcula as forças que serão passadas para os itens imediatamente abaixo no empacotamento e continua a verificação para os demais itens do empacotamento. Caso o centro de massa de algum item não se encontra numa região válida obtemos um empacotamento instável. Depois de verificar todos os itens e constatar que todos se encontram estáveis, conclui-se que o empacotamento obtido está estável.

O problema abordado neste trabalho é resolvido de forma praticamente exata, pois ao

empacotar os itens consideramos apenas as posições inteiras e na verificação da estabilidade analisamos também as posições fracionárias. Para solucionar o problema, desenvolvemos e criamos uma formulação linear que é aplicada para cada instância do problema, obtendo assim um programa linear inteiro que será resolvido pelo pacote *XPress-Optimizer*.

Definimos duas variantes para os problemas de empacotamento em faixa. A primeira variante engloba os problemas para os quais a ordem não é importante, enquanto que na segunda estão os problemas para os quais ela importa. A divisão do problema quanto à ordem não influencia na estabilidade do empacotamento, já que em ambas variantes a solução encontrada retorna um empacotamento estável.

2.3.1 Sem restrição de ordem

Nos problemas sem restrição de ordem, qualquer item pode ficar em qualquer lugar no empacotamento, contanto que o empacotamento obtido seja estável e que a altura do empacotamento seja a menor possível. Formalmente este problema pode ser definido da seguinte forma:

Problema de Empacotamento Estável em Faixa Bidimensional (2EEF): São dados um recipiente $B = (L, \infty)$, de largura L e altura infinita, e uma lista T de n itens ou pequenos retângulos com dimensões (l_i, c_i), para i variando de 1 até n. O objetivo é determinar como empacotar todos os itens da lista T de forma a obter um empacotamento estável minimizando a altura do recipiente.

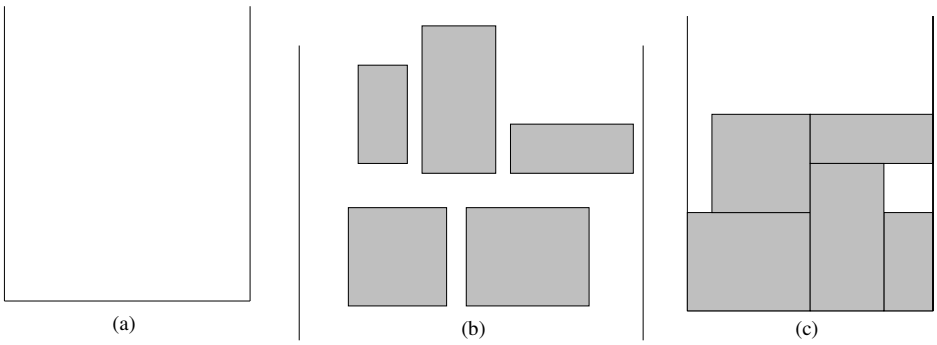


Figura 2.2: Exemplo de um problema 2EEF

Na figura 2.2 temos um exemplo de empacotamento do 2EEF. Em 2.2.(a) temos o recipiente onde os itens serão empacotados. Em 2.2.(b) temos os itens que serão empacotados. Em 2.2.(c) temos um empacotamento estável formado pelos itens dentro deste recipiente.

2.3.2 Com restrição de ordem

Nos problemas com restrições de ordem, os itens devem respeitar a uma ordem estabelecida no empacotamento. A ordem de cada item permite que o empacotamento obtido respeite a prioridade de descarregamento que um item possui sobre o outro. Assim podemos obter um empacotamento onde os itens mais pesados estejam embaixo no empacotamento ou também podemos obter um empacotamento em que os itens que devem ser retirados primeiro do empacotamento estejam localizados nas posições mais altas do empacotamento.

Para acrescentar a restrição de ordem ao problema 2EEF, criamos uma condição que estabelece que um item só pode ser empacotado sobre outro item se a ordem do item que estiver imediatamente acima for " menor ou igual " à ordem do item que se localizar imediatamente abaixo.

Formalmente definimos o 2EEF com restrição de ordem da seguinte forma:

Problema de Empacotamento Estável em Faixa Bidimensional com Restrição de Ordem (2EEFO): São dados um recipiente $B = (L, \infty)$, de largura L e altura infinita, e uma lista T de n itens ou pequenos retângulos com dimensões (l_i, c_i) e ordem o_i, para i variando de 1 até n. O objetivo é determinar como empacotar todos os itens da lista T, satisfazendo a restrição de ordem, de maneira a minimizar a altura do recipiente utilizado e a obter um empacotamento estável.

A restrição de ordem diz que qualquer item i, de ordem o_i, deve se localizar em cima de itens cuja ordem seja maior ou igual a o_i ou deve estar apoiado no fundo do recipiente. Desta maneira, a remoção de um item não será bloqueada por itens de ordem menor.

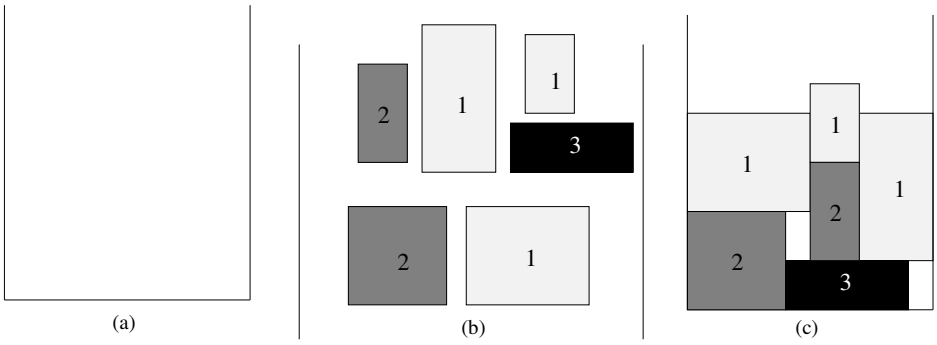


Figura 2.3: Exemplo de uma solução para uma instância do problema 2EEFO.

A figura 2.3 mostra um exemplo de um problema 2EEFO. Em 2.3.(a) temos o recipiente onde os itens serão empacotados. Em 2.3.(b) temos os itens que serão empacotados. Neste

exemplo, os itens que possuem a mesma ordem possuem a mesma cor. O item de cor preta possui ordem 3, o de cor cinza possui ordem 2 e o de cor branca tem ordem 1. Em 2.3.(c) temos um empacotamento formado pelos itens dentro de recipiente. Note que o empacotamento obtido é estável e respeita a restrição de ordem dos itens.

O problema 2EEF é um caso particular do problema 2EEFO. O fato do problema 2EEF não considerar a ordem, equivale resolvermos o problema 2EEFO considerando que a ordem de todos os itens seja igual a 0. Desta forma, no capítulo 4, quando mencionarmos o algoritmo praticamente exato, o domínio do problema e as formulações lineares só abordaremos o problema 2EEFO, uma vez que resolvendo este problema automaticamente estaremos resolvendo o problema 2EEF.

2.4 Considerações Finais do Capítulo

Este capítulo apresentou o problema foco desta dissertação, citando alguns problemas correlatos na revisão literária e descrevendo as duas variantes de empacotamentos estáveis: as que tratam ou não as restrições de ordem.

Capítulo 3

Estabilidade

Este capítulo descreve os procedimentos adotados para verificar a estabilidade estática de um empacotamento qualquer.

Propor meios de verificar o equilíbrio estático de um conjunto de itens não é um procedimento simples, já que algumas suposições devem ser feitas. Além disso, desejamos propor uma metodologia sobre a estabilidade que possa ser facilmente estendida para outros problemas de empacotamento.

Na literatura existem poucos trabalhos que falam sobre a estabilidade em empacotamentos. Em Castro Silva *et al.* [16], encontramos uma heurística para o problema de empacotamento em recipientes tridimensionais. Esta heurística usa uma política de melhor preenchimento possível para um único recipiente junto com a idéia de pontos de canto definido por Martello *et al.* [34].

No trabalho de Junqueira *et al.* [29] as estabilidades estática e dinâmica são tratadas com relação à proporção da área de contato ocupada entre a face inferior de uma caixa e a face superior de uma ou mais caixas que estejam imediatamente abaixo dela. Esta proporção é configurada por um parâmetro de estabilidade que varia de 0 a 1. Quando este parâmetro de estabilidade for igual a 1 significa que as faces inferiores de todas as caixas devem estar 100% suportadas pelas faces superiores de uma ou mais caixas colocadas imediatamente abaixo delas. No outro extremo, quando o parâmetro valer 0, indica que não há exigências quanto à estabilidade das caixas em relação ao eixo z (por exemplo, as caixas podem estar apenas parcialmente apoiadas, ou mesmo "flutuando"dentro do contêiner).

Adiante, apresentaremos o estudo da estabilidade dentro de problemas de empacotamento. Também, descrevemos em alto nível um algoritmo capaz de verificar se um dado empacotamento é estaticamente estável.

3.1 Verificando a Estabilidade

Para lidar com a estabilidade dos itens no empacotamento, fazemos uso dos conceitos de resistência dos materiais e equilíbrio estático de corpos rígidos. Deste modo, usaremos os conceitos físicos relativo à Força, Centro de Gravidade, Centro de Massa, Momento de uma força, Momento Fletor, Vigas e a equação dos Três Momentos [8, 24, 25, 36].

Sem perda de generalidade, associamos os itens a elementos estruturais chamados de *vigas*. O objetivo é utilizar as formulações e conceitos bem definidos pela literatura [5, 21, 40] para tais tipos de elementos. Deste modo, não faremos distinção entre um item retangular e uma viga.

Inicialmente, *força* é a ação de um corpo sobre outro corpo, caracterizada como uma grandeza vetorial. Vamos supor que existe apenas a ação da força peso, isto é, lidaremos apenas com o peso dos itens de modo que qualquer outra força será desconsiderada. Note que podemos assumir outras forças externas, como a ação do vento ou, no caso do transporte de cargas, a ação dinâmica do movimento do caminhão sobre a carga.

A força peso é uma força de atração gravitacional e está distribuída sobre toda a extensão do item. Esta força pode ser substituída por uma única resultante que atua no centro de massa do item. Por tratarmos do equilíbrio estático dos corpos rígidos as forças não variam com o tempo e por utilizarmos conceitos de vigas os itens podem apresentar alguma flexibilidade.

Consideramos que toda a massa do item está concentrada em um único ponto chamado *centro de massa*. Assim, considerando que o campo gravitacional é uniforme, o centro de massa é o ponto onde atua a resultante da força peso daquele item. A equação (3.1) permite calcular o centro de massa de um item que sofre a ação de uma ou mais forças resultantes.

$$\overrightarrow{C} = \frac{\sum_{i=1}^{n} \overrightarrow{r_i} f_i}{\sum_{i=1}^{n} m_i} \quad \Rightarrow \quad \begin{array}{l} C_x = \frac{\sum_{i=1}^{n} r_i^x f_i}{\sum_{i=1}^{n} f_i} \\ \\ C_y = \frac{\sum_{i=1}^{n} r_i^y f_i}{\sum_{i=1}^{n} f_i} \end{array} ; \tag{3.1}$$

sendo que o vetor $\overrightarrow{C}$, com as componentes C_x e C_y, indica o centro de massa do item; f_i corresponde a i-ésima força resultante aplicada sobre o item e cuja distância a um referencial inercial adotado (Primeira Lei de Newton) é descrita pelo vetor $\overrightarrow{r_i}(r_i^x,\ r_i^y)$. O canto inferior esquerdo do item foi adotado como o referencial inercial. Referencial inercial é um referencial para o qual se uma partícula (item) não está sujeita a forças, então está parada ou em movimento retilíneo uniforme.

Na equação (3.1), caso um item i de dimensões (l_i, c_i) sofra apenas a atuação de sua força peso F_i, seu centro de massa pode ser obtido diretamente e corresponde à posição $(\frac{l_i}{2}, \frac{c_i}{2})$.

Agora, podemos dizer que um item estará em equilíbrio estático (ou é estável) quando seu centro de massa estiver nas seguintes *regiões estáveis*: ou sobre algum dos seus itens adjacentes; ou no meio de dois dos seus itens adjacentes; ou em contato direto com a superfície do recipiente. Em qualquer outro caso, o centro de massa estará em uma região não estável de modo que o item irá rotacionar e, consequentemente, desmanchar o empacotamento feito. Ao mencionar itens adjacentes, estamos nos referindo aos itens imediatamente abaixo e em contato direto com o item atual. Note que desprezamos as laterais do recipiente, tais que estas não impedirão qualquer item de rotacionar.

Ao supor que todos os itens são homogêneos e de mesmo material, temos o módulo da força peso para um item i de dimensões (l_i, c_i) calculado por:

$$P_i = c_i l_i g; \tag{3.2}$$

onde g é a aceleração da gravidade, por exemplo $g = 9{,}8\ m/s^2$. Assumimos que a massa do item é proporcional a sua área.

Desta forma, para verificar a estabilidade de um empacotamento qualquer, a idéia é analisar item a item de modo a obter o seu centro de massa e verificar se tal centro está em uma região estável. Caso isto seja verdade para o item atual, as forças que atuam no item serão propagadas para seus itens adjacentes. Por outro lado, caso o centro de massa esteja em uma região não estável, o empacotamento será dado como instável.

Para determinar o quanto de força está sendo passada para os itens adjacentes, aplicamos as equações de equilíbrio estático para um corpo rígido considerando o conjunto de forças que atuam no item atual e em seus adjacentes. Então, um item está em equilíbrio quando as resultantes do sistema de forças e dos momentos que atuam nele são iguais a zero, a saber:

$$\sum_{i=0}^{n} \overrightarrow{F_i} = \overrightarrow{0}, \qquad \sum_{i=0}^{n} \overrightarrow{M_i^O} = \overrightarrow{0}; \tag{3.3}$$

tal que $\sum_{i=0}^{n} \overrightarrow{F_i}$ e $\sum_{i=0}^{n} \overrightarrow{M_i^O}$ representam, respectivamente, a soma vetorial de todas as forças e momentos que atuam no item i. O canto inferior esquerdo do item foi adotado como o referencial inercial O durante o cálculo do momento.

Por estarmos lidando com o caso bidimensional e não ter qualquer outra força além do próprio peso do item cuja direção é vertical para baixo, somente a componente y da

força e z do momento precisam ser consideradas:

$$\sum_{i=0}^{n} F_{iy} = 0, \qquad \sum_{i=0}^{n} M_{iz}^{O} = 0. \tag{3.4}$$

Sabendo que apenas a força peso dos itens será considerada para a análise, torna-se necessário determinar como propagá-la para seus itens adjacentes, além de determinar o local onde tal força irá atuar nestes itens. Para tanto, temos três casos a considerar:

- (i) o item atual possui apenas um item adjacente;
- (ii) o item atual possui exatamente dois itens adjacentes, e;
- (iii) o item atual possui três ou mais itens adjacentes.

Decidimos dividir em três casos, pois os casos (i) e (ii) foram bem frequentes nos testes computacionais realizados, além de serem casos bases para o terceiro caso, que é o mais geral. Outro fato, é que estes dois casos possuem formulações diretas de modo a agilizar a execução da rotina que verifica a estabilidade.

Sabendo que vigas são elementos estruturais que sofrem a ação de forças/cargas, de modo que podem se deformar e/ou se deslocar da posição original, vamos usar a *hipótese de pequenos deslocamentos*. Isto significa que as condições de equilíbrio são impostas para a geometria original (indeformada) da estrutura [5, 8].

3.1.1 Caso (i)

O caso (i) é o mais simples e consiste em um ou mais itens cada um sobre exatamente outro item. Deste modo, não é necessário recorrer à idéia de vigas para analisar este caso.

Seja i o item que sofre apenas a atuação de sua própria força peso $\overrightarrow{P_i}$ no seu centro de massa $\overrightarrow{C_i}$ e cujo item adjacente é o item j. Assuma também que o item j possua apenas o item i imediatamente acima dele e que o centro de massa do item i está em uma região estável. A figura 3.1 retrata esta situação.

De acordo com a terceira Lei de Newton, o item i sofrerá apenas a atuação da força $\overrightarrow{P_i}$, a qual é passada no ponto $\overrightarrow{C_i}$. Posteriormente, ao analisar o item j, nele estará atuando a sua própria força peso $\overrightarrow{P_j}$ no seu centro de massa $\overrightarrow{C_j}$ e as forças que vieram dos itens imediatamente acima. Neste caso, apenas a força peso do item i.

Como o item j possui mais de uma força atuante, seu novo centro de massa deve ser calculado, conforme a equação (3.1). Se este novo centro de massa estiver em uma região estável, o item j é dado como estável e a força resultante que atua nele é propagada para os seus itens adjacentes (conforme um dos três casos). Por outro lado, o item j é dado como instável e a rotina que verifica a estabilidade finaliza sua execução.

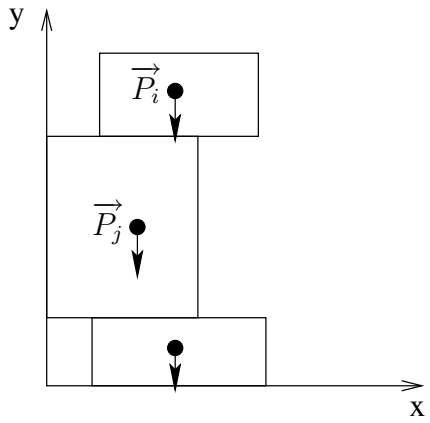


Figura 3.1: Representação do caso (i).

3.1.2 Caso (ii)

O caso (ii) trata o caso em que um item possui exatamente dois itens adjacentes. A representação através de vigas neste contexto simplifica a análise a ser feita. Por exemplo, considere que o item a ser analisado é o item k da figura 3.2. Neste item temos a atuação de sua própria força peso no centro de massa $\overrightarrow{C_k}$, de duas outras forças: $\overrightarrow{Fp_i}$, aplicada no ponto $\overrightarrow{p_i}$, e $\overrightarrow{Fp_j}$, aplicada em $\overrightarrow{p_j}$; e os itens k_1 e k_2 que são adjacentes ao item k. Representando o item k através de vigas, obtemos 3 vigas. A primeira vai do início do item k até a força de reação R_1, a segunda fica entre as forças de reação R_1 e R_2 e a terceira vai da força R_2 até o final do item k. A figura 3.2 apresenta um exemplo de configuração para o caso 3.1.2 usando itens retangulares e uma analogia através de vigas.

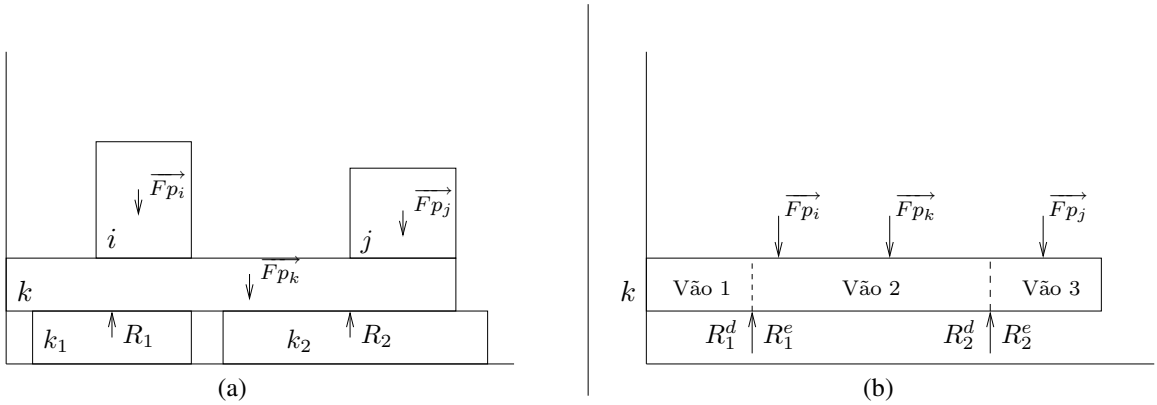


Figura 3.2: Configuração do caso (ii) usando os itens (a) e a representação através de vigas (b).

Assumindo que o novo centro de massa para o item k está em uma região estável, o objetivo é, então, determinar como propagar as forças para seus itens adjacentes.

Observe que, conforme mostra a figura 3.2, o item k representa a viga e os seus itens adjacentes, k_1 e k_2, consistem nos apoios (elementos estruturais conhecidos como pilares) que interagem com a viga. Note que nesta interação surgem duas forças, R_1 e R_2, chamadas de forças de reação, que são as forças que os apoios k_1 e k_2 exercem sobre a viga.

A **viga** é uma estrutura formada por uma barra, de eixo plano, submetida a esforços contidos no plano da estrutura. Consideramos este plano como sendo o sistema de coordenadas xy. Os esforços considerados são o peso próprio da viga, item 3 (F_3), as forças que vem dos itens imediatamente acima (em contato direto), itens 4 e 5 (F_4 e F_5), e as forças de reação (força que os apoios, itens 1 e 2, exercem sobre a viga), R_1 e R_2, como pode ser visto na figura 3.3.

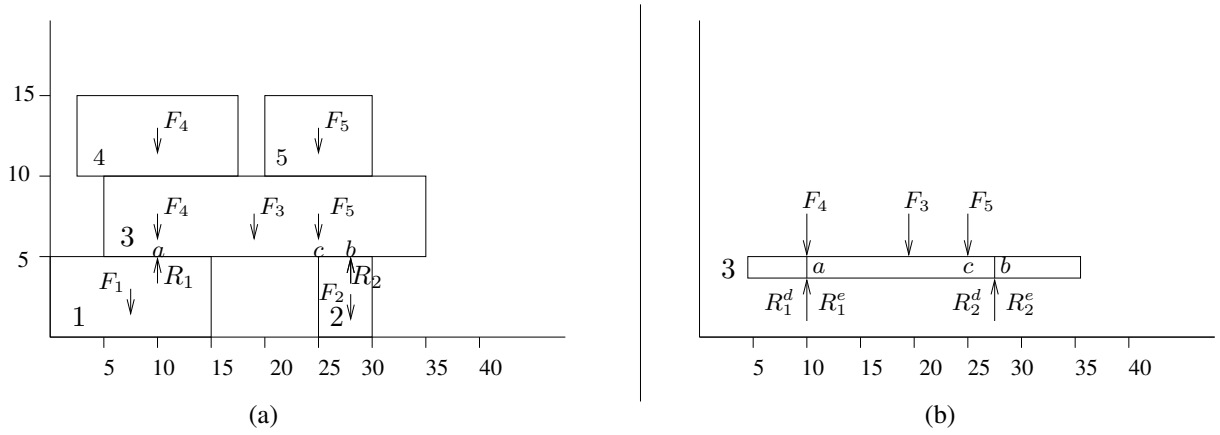


Figura 3.3: a) Exemplo de um diagrama de corpo livre para o caso 2. (b) Modelagem através de uma viga (item 3) com as forças (que vem dos itens acima em contato direto) e as reações de apoio (itens adjacentes).

Momento fletor em uma seção (em um ponto) é a soma algébrica dos momentos produzidos pelos esforços externos (peso próprio, forças oriundas dos itens acima em contato direto e reações de apoio) aplicadas transversalmente ao eixo longitudinal da viga (eixo x). Produzindo, assim, um esforço que tende a curvar o eixo longitudinal para cima (tração) ou para baixo (compressão).

Denominamos por força resultante, o conjunto de todas as forças que atuam na viga,

exceto as de reação. Então, para satisfazer a terceira Lei de Newton e, consequentemente, as equações de equilíbrio, a força resultante precisa ser repartida entre todos os apoios em contato direto com a viga. Sem perda de generalidade, assumimos que o local onde a força resultante da viga age nos apoios corresponde ao ponto médio da extensão de contato entre a viga e o respectivo apoio, conforme apresentado na figura 3.2.(a) e na figura 3.3.(a).

Como o objetivo é determinar as forças que passam do item atual para os itens adjacentes, ou seja, calcular as forças de reação R_1 e R_2, a análise consiste em dividir a viga em três partes, denominadas de vãos: o vão 1, a esquerda de R_1; o vão 2 entre R_1 e R_2; e, o vão 3 a direita de R_2.

Dessa forma, aplicamos as equações de equilíbrio (3.3), realizando os cálculos inicialmente para os vãos da extremidade. Neste caso, os vãos 1 e 3. Assim, obtemos os momentos e uma parcela das reações de apoio, R_1^d e R_2^e. Em seguida, aplicamos a equação (3.3) nos vãos intermediários (neste caso, somente o vão 2), obtendo assim a outra parcela que compõem as forças de reação, R_1^e e R_2^d. Por fim, utilizando o *princípio da superposição* [24], obtemos os valores finais das forças de reação nos apoios, que correspondem aos valores das forças que passam para os respectivos itens adjacentes.

A figura 3.4 mostra o exemplo de um diagrama de corpo livre para cada vão da viga da figura 3.3.

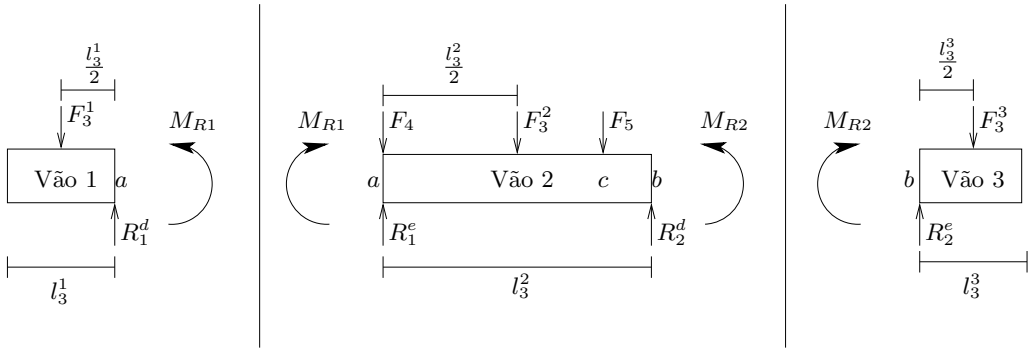


Figura 3.4: Diagrama de corpo livre para cada vão da viga da figura 3.3.

Abaixo temos um exemplo dos cálculos realizados em cada vão da figura 3.4.

Vão 1:

$$\begin{aligned}
\sum M^a_{R1}{=}0 \;&\Rightarrow\; (F^1_3 \times \frac{l^1_3}{2}) + M_{R1} + R^d_1 \times 0 = 0 \\
&\Rightarrow\; (5 \times 5 \times 9,8) \times \frac{5}{2} + M_{R1} = 0 \\
&\Rightarrow\; M_{R1} = -612,5
\end{aligned} \tag{3.5}$$

$$\begin{aligned}
\sum F_y{=}0 \;&\Rightarrow\; R^d_1 - F^1_3 = 0 \\
&\Rightarrow\; R^d_1 - 245 = 0 \\
&\Rightarrow\; R^d_1 = 245(N)
\end{aligned}$$

Vão 3:

$$\begin{aligned}
\sum M^b_{R2}{=}0 \;&\Rightarrow\; -(F^3_3 \times \frac{l^3_3}{2}) - M_{R2} + R^d_2 \times 0 = 0 \\
&\Rightarrow\; -(5 \times 7,5 \times 9,8) \times \frac{7,5}{2} - M_{R2} = 0 \\
&\Rightarrow\; M_{R2} = -1378,125
\end{aligned} \tag{3.6}$$

$$\begin{aligned}
\sum F_y{=}0 \;&\Rightarrow\; R^e_2 - F^3_3 = 0 \\
&\Rightarrow\; R^e_2 - 367,5 = 0 \\
&\Rightarrow\; R^e_2 = 367,5(N)
\end{aligned}$$

Vão 2:

$$\begin{aligned}
\sum M^b_{R2}{=}0 \;&\Rightarrow\; (F^2_3 \times \frac{l^2_3}{2}) + F_4 \times l^2_3 + F_5 \times (b{-}c) + M_{R2} + R^d_2 \times 0 - R^e_1 \times l^2_3 - M_{R1} = 0 \\
&\Rightarrow\; (5 \times 17,5 \times 9,8) \times \frac{17,5}{2} + 735 \times 17,5 + 490 \times 2,5 + \\
&\quad\; (-1378,125) - R^e_1 \times 17,5 - (-612,5) = 0 \\
&\Rightarrow\; (7503,125 + 12862,5 + 1225 + (-1378,125) - R^e_1 \times 17,5 - (-612,5) = 0 + \\
&\Rightarrow\; R^e_1 = 1190
\end{aligned} \tag{3.7}$$

$$\begin{aligned}
\sum F_y{=}0 \;&\Rightarrow\; R^d_2 + R^1_2 - F_4 - F^2_3 - F_5 = 0 \\
&\Rightarrow\; R^d_2 - 1190 - 735 - 857,5 - 490 = 0 \\
&\Rightarrow\; R^d_2 = 892,5(N)
\end{aligned}$$

Então, temos como forças de apoio resultante:

$$\begin{aligned} R_1 &= R_1^d + R_1^e = 245 + 1190 = 1435 \\ R_2 &= R_2^e + R_2^d = 367,5 + 892,5 = 1260 \end{aligned} \tag{3.8}$$

Para confirmar o resultado, basta aplicar $\sum F_y = 0$ considerando toda a viga, a saber:

$$\begin{aligned} \sum F_y = 0 \;&\Rightarrow\; R_1 + R_2 - F_4 - F_3 - F_5 \\ &\Rightarrow\; 1435 + 1260 - 735 - 1470 - 490 = 0 \\ &\Rightarrow\; 2695 - 2695 = 0 \end{aligned} \tag{3.9}$$

3.1.3 Caso (iii)

O último caso é uma extensão natural do caso (ii). A diferença é que teremos três ou mais itens adjacentes. Assim, a viga a ser analisada possui três ou mais apoios, caracterizando uma *viga hiperestática*. Nesta estrutura o número de incógnitas (forças de reação, no nosso caso) a terem seu valor calculado é maior que o número de equações (de equilíbrio) disponíveis.

As equações de equilíbrio que apresentamos em (3.3) permitem resolver diretamente apenas os casos (i) e (ii), já que temos somente duas equações. Para o caso (iii), faz-se necessário determinar os momentos (internos) que aparecem nos apoios intermediários, isto é, em cada reação de apoio. Conhecendo o momento em cada um destes pontos, torna-se possível dividir a viga em vãos independentes e, então, aplicar as equações de equilíbrio em cada vão para, por fim, fazer uso do princípio da superposição e obter as forças (de reação) que atuam nos itens adjacentes.

Um dos métodos existentes para lidar com estruturas hiperestáticas e, então, a obter as forças de reação nos apoios, denomina-se *método da Equação dos Três Momentos* [8, 19]. Este método fornece um equacionamento simples que permite calcular o momento fletor em cada reação de apoio da viga (momentos hiperestáticos de uma viga contínua). Conhecendo o momento fletor em cada um destes pontos, torna-se possível tratar a viga como vãos independentes, tal que se torna possível a aplicação das equações de equilíbrio dos corpos rígidos vistas na equação (3.3). A figura 3.5.(a) mostra um exemplo de uma configuração que recai sobre o caso 3 e a figura 3.5.(b) mostra a representação esquemática desta configuração através de uma viga com suas devidas forças e apoios.

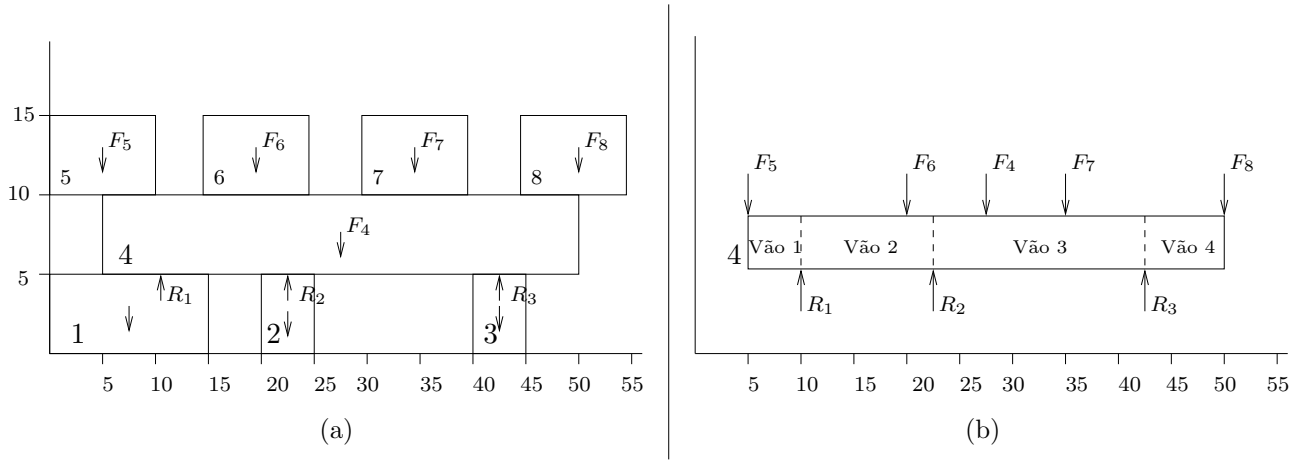


Figura 3.5: (a) Configuração do caso 3. (b) Representação do caso (a) através de uma viga com suas devidas forças e apoio.

A aplicação do método da Equação dos Três Momentos consiste em considerar trechos subsequentes de uma viga contínua que contenha dois vãos (seja os vãos n e $n+1$) e três apoios (sejam os apoios R_{n-1}, R_n, R_{n+1}, além dos momentos fletores nos respectivos apoios, M_{n-1}, M_n, e M_{n+1}), conforme mostra a figura 3.6. Note que neste trecho pode haver a atuação de forças, como o próprio peso e as forças resultantes que vem dos itens acima em contato direto.

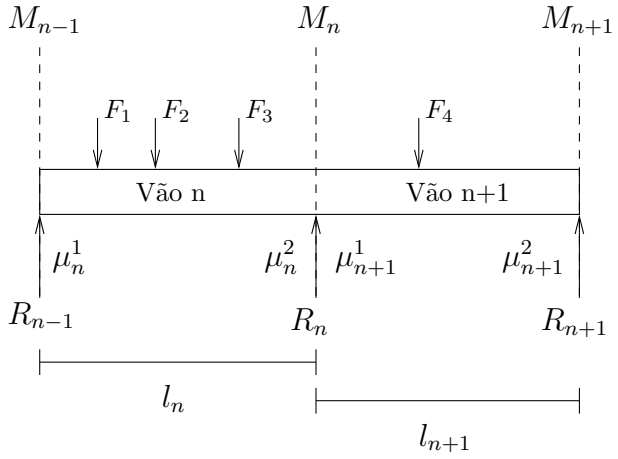


Figura 3.6: Trecho de uma viga contínua com dois vãos e três apoios.

Os itens do problema possuem uma forma geométrica constante, ou seja, são retângulos. Neste caso, aproveitamos que o momento de inércia é constante em todos

os vãos e de vão para vão, e assim obtemos a seguinte equação dos três momentos:

$$l_n \times M_{n-1} + 2 \times (l_n + l_{n+1}) \times M_n + l_{n+1} \times M_{n+1} = -6 \times (\mu_n^2 + \mu_{n+1}^1). \qquad (3.10)$$

sendo l_n e l_{n+1} a largura dos vãos n e $n+1$, respectivamente; μ_n^2 e μ_{n+1}^1 os fatores de carga, respectivamente, nos vãos n, apoio da direita, e $n+1$, apoio da esquerda. Os fatores de carga correspondem ao somatório de todas as forças (exceto as de reações) que atuam num determinado vão.

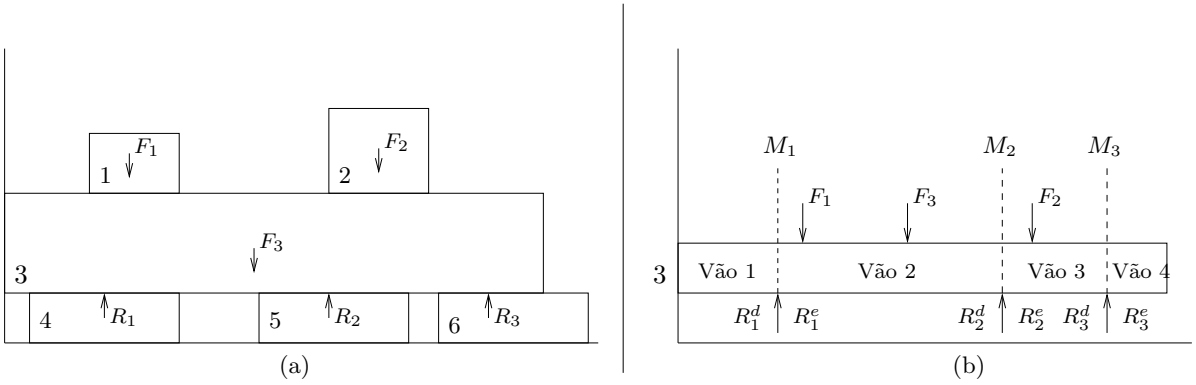


Figura 3.7: Exemplo do caso (iii) para um item com três itens adjacentes.

Especialmente para o nosso caso, uma viga com n vãos possui exatamente $n-1$ apoios, já que os vãos extremos, vão 1 e vão n estão em balanço. Estes vãos não são considerados durante a aplicação da equação dos Três Momentos, pois os momentos, M_1 e M_{n-1}, nos respectivos apoios, 1 e $n-1$, são obtidos diretamente pela equação (3.3). Além disso, as parcelas das forças de reação, R_1^d e R_{n-1}^e, também são obtidas diretamente. A aplicação do método é feita considerando vãos subsequentes dois a dois, começando pelo vão 2 até o vão $n-1$. Para mais detalhes veja [20, 6]. A figura 3.7 representa esquematicamente um item 3 que possui três itens adjacentes, itens 4, 5 e 6. Note que temos 4 vãos e 3 apoios.

A figura 3.8 mostra uma viga com 6 vãos. Desconsiderando os vãos em balanço, temos uma viga com 4 vãos e aplicamos 3 vezes a equação dos três momentos para calcular os momentos fletores. Resultando assim num sistema linear de 3 equações e 3 incógnitas (M_2, M_3 e M_4), já que M_1 e M_5 são previamente conhecidos. Em (3.11) temos o conjunto de equações deste sistema linear.

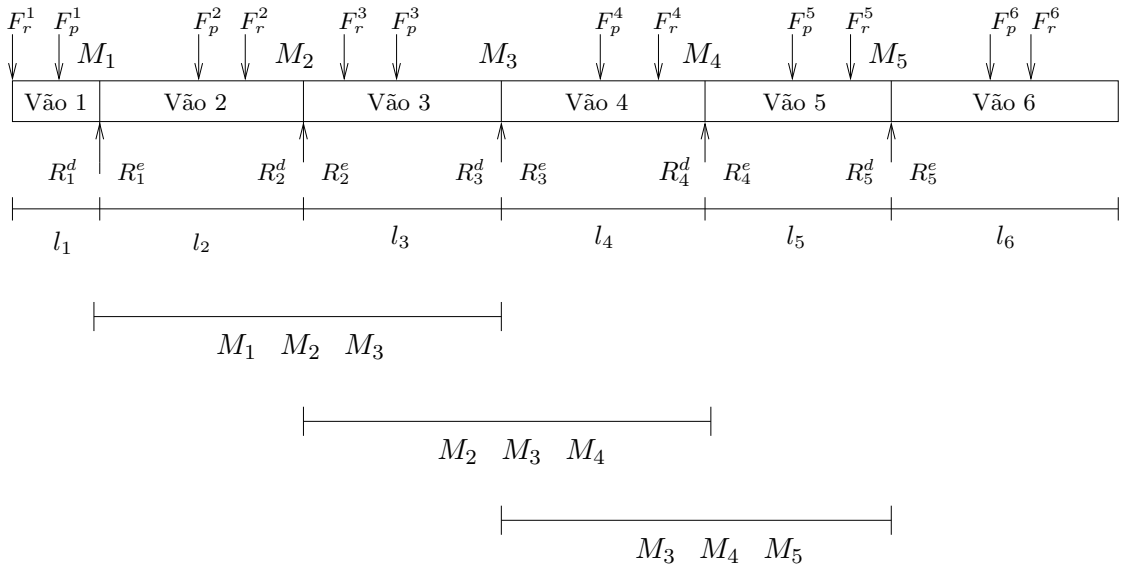


Figura 3.8: Diagrama exemplificando a aplicação sucessiva da equação dos 3 momentos para uma viga com 5 apoios.

$$
\begin{aligned}
M_1 &= -(F_p^1 \times \tfrac{l_1}{2}) - \textstyle\sum_{i=1}^{r_1} F_i^1 \times d_i^1 \\
l_2 \times M_1 + 2 \times (l_2 + l_3) \times M_2 + l_3 \times M_3 &= -6 \times (\mu_2^2 + \mu_3^1) \\
l_3 \times M_2 + 2 \times (l_3 + l_4) \times M_3 + l_4 \times M_4 &= -6 \times (\mu_3^2 + \mu_4^1) \\
l_4 \times M_3 + 2 \times (l_4 + l_5) \times M_4 + l_5 \times M_5 &= -6 \times (\mu_4^2 + \mu_5^1) \\
M_5 &= -(F_p^6 \times \tfrac{l_6}{2}) - \textstyle\sum_{i=1}^{r_6} F_i^6 \times d_i^6
\end{aligned}
\tag{3.11}
$$

Apresentamos adiante a formulação matricial do método da equação dos Três Momentos para o caso em que temos uma viga com n vãos e $n-1$ apoios. Fazendo a analogia, temos um item com $n-1$ itens adjacentes. Observe que em cada vão da viga atuam forças, como sua própria força peso e as forças resultantes que vem dos itens acima em contato direto. Logo, precisamos resolver o seguinte sistema:

$$\mathbf{F}\ \mathbf{m} = \mathbf{d}, \tag{3.12}$$

em que $\mathbf{F}$ é denominada matriz de flexibilidade, sendo tri-diagonal e simétrica; $\mathbf{m}$ é o vetor de momentos, que se deseja obter; e, $\mathbf{d}$ é o vetor de forças/cargas aplicadas no item em análise.

Em (3.13) temos a representação matricial da figura 3.8 .

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ l_2 & 2\times(l_2+l_3) & l_3 & 0 & 0 \\ 0 & l_3 & 2\times(l_3+l_4) & l_4 & 0 \\ 0 & 0 & l_4 & 2\times(l_4+l_5) & l_5 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \end{bmatrix} = \begin{bmatrix} -(F_p^1 \times \frac{l_1}{2}) - \sum_{i=1}^{r_1} F_i^1 \times d_i^1 \\ -6\times(\mu_2^2+\mu_3^1) \\ -6\times(\mu_3^2+\mu_4^1) \\ -6\times(\mu_4^2+\mu_5^1) \\ -(F_p^6 \times \frac{l_6}{2}) - \sum_{i=1}^{r_6} F_i^6 \times d_i^6 \end{bmatrix} \quad (3.13)$$

Iniciando no vão 2 e indo até o vão $n-1$, assumindo que o apoio j está entre os vãos j e $j+1$, as três colunas não nulas ($j-1$, j e $j+1$) na linha j ($j = 2, \ldots, n-2$) da matriz **F** são calculados como:

$$\begin{aligned} F_{j,j-1} &= l_j \\ F_{j,j} &= 2(l_j + l_{j+1}), \\ F_{j,j+1} &= l_{j+1} \end{aligned} \quad (3.14)$$

sendo l_j a respectiva largura do vão j. Por outro lado, o vetor **d**, na posição j, corresponde a:

$$d_j = -6(\mu_j^2 + \mu_{j+1}^1), \quad (3.15)$$

em que μ_j^2 e μ_{j+1}^1 representam os fatores de carga do vão j, para o apoio da direita (índice 2 no expoente), e do vão $(j+1)$, para o apoio da esquerda (índice 1 no expoente), respectivamente.

Os fatores de carga são computados em função das forças (exceto as de reação) que atuam em cada vão. Quando existir mais de uma força, caso em que há a força peso do vão e as forças resultantes que vêm dos itens acima em contato direto com aquele vão, os fatores de carga finais são dados pela soma dos fatores de carga obtidos individualmente para cada força. A figura 3.9 mostra as duas formas de calcular os fatores de carga: (a) quando a força é distribuída, caso da força peso do item; (b) quando a força é concentrada, caso das forças resultantes que vem dos itens acima em contato direto.

Os demais valores, $F_{1,1}$, $F_{n-1,n-1}$, d_1 e d_{n-1} são obtidos diretamente da seguinte forma:

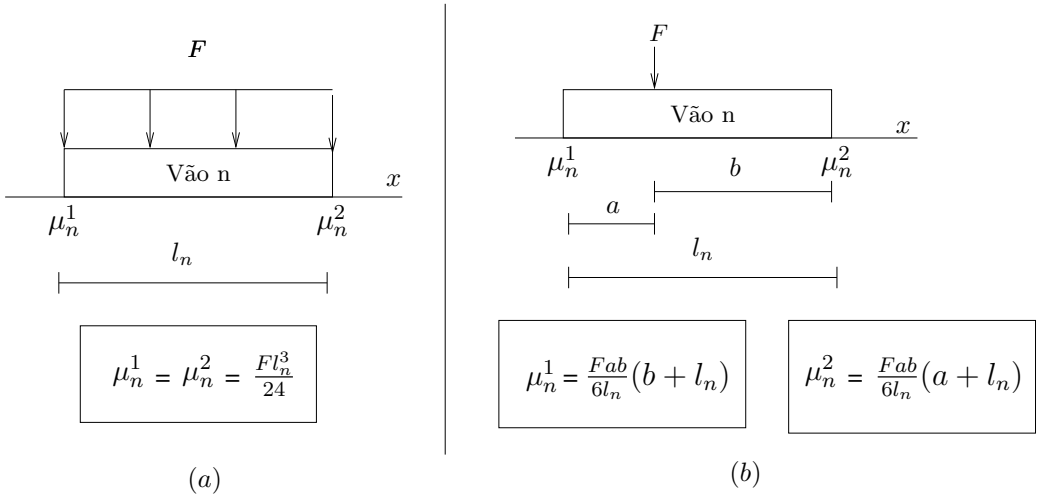


Figura 3.9: Fórmula para obter os fatores de carga: (a) para uma força distribuída; (b) para uma força concentrada.

$$\begin{aligned} F_{1,1} &= 1 \\ F_{n-1,n-1} &= 1 \\ d_1 &= -(F_p^1 \frac{l_1}{2}) - \sum_{i=1}^{r_1} F_i^1 d_i^1 \\ d_{n-1} &= -(F_p^{n-1} \frac{l_{n-1}}{2}) - \sum_{i=1}^{r_{n-1}} F_i^{n-1} d_i^{n-1}, \end{aligned} \tag{3.16}$$

em que F_p^1 e l_1 representam, respectivamente, a força peso e a largura do vão 1; r_1 é a quantidade de forças oriundas dos itens acima em contato direto que atuam no vão 1 (exclui-se a própria força peso do vão), sendo F_i^1 o módulo da i-ésima força e d_i^1 sua distância em relação ao referencial inercial adotado; O mesmo se aplica para F_p^{n-1}, l_{n-1}, r_{n-1}, F_i^{n-1} e d_i^{n-1}, porém considerando o vão $n-1$.

Note que agora basta resolver a equação (3.12) para obter os momentos nos apoios. A partir daí, podemos aplicar as equações de equilíbrio em cada vão e obter as forças (de reação) passadas para os itens adjacentes.

3.1.4 Algoritmo para Verificar a Estabilidade

A partir dos três casos discutidos anteriormente, temos informações suficientes para determinar as forças que são passadas para cada item e, então, verificar a estabilidade de um empacotamento bidimensional. O Algoritmo 1 descreve, em alto nível, a rotina que verifica se um determinado empacotamento é estável.

A complexidade de tempo, no pior caso, do Algoritmo 1 é $O(n^4)$, sendo n a quantidade de itens dentro do empacotamento I. Os passos descritos nas linhas $1.1 - 1.8$ foram efetuados em tempo $O(n \log n)$, dado a utilização de um algoritmo de ordenação dos itens. O laço das linhas $1.9 - 1.30$ consome tempo $O(n^4)$. Note que na linha 1.35, durante a análise do caso (iii), é preciso calcular a inversa da matriz de flexibilidade $\mathbf{F}$, cujo tempo estimado é $O(n^3)$.

3.2 Considerações Finais do Capítulo

Neste capítulo mostramos como verificar a estabilidade estática de um empacotamento. Abordamos de forma simples e objetiva os conceitos físicos que a envolve. Descrevemos o algoritmo que verifica o equilíbrio de um dado empacotamento de itens.

Além disto, usamos uma metodologia que analisa a estabilidade de forma diferente das outras abordagens usadas na literatura para problemas de Corte e Empacotamento. Esta metodologia trata a estabilidade de forma exata, sendo similar ao que acontece na prática e faz com que seja bem simples adicionar outras fontes de força no empacotamento, como é o caso de forças que podem atuar nos itens (a ação do vento ou a inclinação do recipiente).

Algoritmo 1: Rotina que verifica a estabilidade de um empacotamento bidimensional qualquer.

Entrada: Empacotamento I de um conjunto de n itens retangulares.

Saída: Retorna se I é estável ou não.

Seja $p_i(x_i, y_i)$ o canto inferior esquerdo dentro do empacotamento, para cada $i \in I$.
Seja $S(I)$ uma ordenação dos itens de forma não-crescente em y_i e para itens com mesma altura, de maneira crescente em x_i.
Seja $A(i)$ o conjunto de itens adjacentes a i, para cada item $i \in S(I)$.
para cada $i \in S(I)$ **faça**
    para cada $j \in A(i)$ **faça**
        Seja I_{ij} o segmento que representa a extensão de contato entre o item i e j.
        Seja (x^s_{ij}, y^s_{ij}) e (x^e_{ij}, y^e_{ij}) os pontos inicial e final de I_{ij}, respectivamente.
        Seja $(x^p_{ij}, y^p_{ij}) = (\frac{(x^e_{ij}-x^s_{ij})}{2}, \frac{(y^e_{ij}-y^s_{ij})}{2})$ o ponto médio de I_{ij}.
para cada $i \in S(I)$ **faça**
    se *i NÃO recebeu forças vindas de itens acima em contato direto* **então**
        Obter o Centro de Massa $\overrightarrow{CM_i}$ do item i.
        se $\overrightarrow{CM_i}$ *está em uma região estável* **então**
            Item i *é estável*. **CONTINUE**.
        senão
            devolva Item i está instável. Empacotamento instável.
    senão
        Obter o Centro de Massa $\overrightarrow{CM_i}$ do item i, vide equação (3.1), conforme as forças que vêm dos itens acima em contato direto.
        se $\overrightarrow{CM_i}$ *está em uma região estável* **então**
            Item i *é estável*. **CONTINUE**.
        senão
            devolva Item i está instável. Empacotamento instável.
    Seja $k \leftarrow |A(i)|$, a quantidade de itens adjacentes ao item i.
    Obter (e armazenar) as forças que passam de i para seus adjacentes em $A(i)$.
    se $k = 1$ **então**
        Ver **caso (i)**.
    senão
        se $k = 2$ **então**
            Ver **caso (ii)**.
        senão
            Ver **caso (iii)**.
devolva Empacotamento estável.

Capítulo 4

Modelo de Programação Linear Inteira

Neste capítulo será apresentado o domínio do problema, duas formulações lineares e inteiras que modelam o problema de empacotamento em faixa bidimensional (2EEFO) e descreveremos o algoritmo proposto e utilizado para resolver o 2EEFO .

Na seção 4.1 explicaremos porque optamos por um domínio de pontos discretizados, na seção 4.2 apresentaremos as formulações explicando suas variáveis, restrições e funções objetivos e na seção 4.3 será abordado o algoritmo que resolve o 2EEFO .

4.1 Domínio do Problema

Para representar computacionalmente o problema, optamos por um domínio discretizado, isto é, um domínio formado por um conjunto de pontos finitos. Desta forma o recipiente e os itens, que são exemplos de uma região contínua, serão discretizados em um conjunto (ou malha) de pontos.

Para discretizar um recipiente de dimensões $L \times C$, onde L é a largura e C a altura, basta transformar estas dimensões num conjunto de pontos inteiros. A equação (4.1) mostra este conjunto. Nesta equação, X é o conjunto de pontos inteiros obtidos para a largura do recipiente e Y é o conjunto de pontos inteiros para a altura do recipiente.

$$\begin{aligned} X &= \{x \in \mathbb{N} \setminus x \in [0, L-1]\}, \\ Y &= \{y \in \mathbb{N} \setminus y \in [0, C-1]\}. \end{aligned} \tag{4.1}$$

Para diminuir o tamanho do conjunto de pontos, podemos escolher, sem perda de generalidade, só os pontos do conjunto Y que fazem parte da combinação linear das alturas dos itens que devem ser empacotados.

Com os conjuntos X e Y definidos, obtemos $\mathcal{P}$ que é o conjunto formado por todos os pontos da malha. Na equação (4.2) temos a definição formal do conjunto $\mathcal{P}$.

$$\mathcal{P} \ = \ \{(x, y) \in \mathbb{N}^2 \setminus x \in X \ e \ y \in Y\}. \tag{4.2}$$

A figura 4.1 representa a malha de pontos para um recipiente de dimensões 4×4 com os conjuntos X e Y. Perceba que nesta figura a última linha e a última coluna do recipiente estão tracejadas. Os pontos que fazem parte desta linha e coluna não são representados no modelo, porque como os itens possuem dimensões inteiras, não existe nenhum item que faça parte de um empacotamento viável quando for empacotado em algum destes pontos.

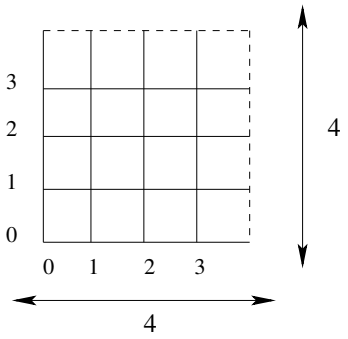


Figura 4.1: Malha de pontos para um recipiente 4×4

A discretização dos itens é semelhante à discretização do recipiente. Cada item i do empacotamento possui uma largura l_i e uma altura c_i. Com a largura l_i são obtidos os pontos que se localizam no eixo horizontal e com a altura c_i os pontos do eixo vertical. Seja I_i o conjunto dos pontos discretizados do item i, onde i varia de 1 até n, sendo n a quantidade de itens do empacotamento. A figura 4.2 mostra o item x de dimensões 3×3, sendo I_x o conjunto de pontos discretizados para este item.

O conjunto I_x, desde que respeite as condições de empacotamento, pode ser colocado em qualquer ponto do recipiente. Para empacotar I_x adotamos como referência o ponto correspondente ao seu canto inferior esquerdo. Na figura 4.2 este ponto é representado pelo ponto branco.

Com a discretização do recipiente e dos itens em conjuntos de pontos, o objetivo do problema passa a ser encontrar quais os pontos do conjunto $\mathcal{P}$ em que cada conjunto I_i,

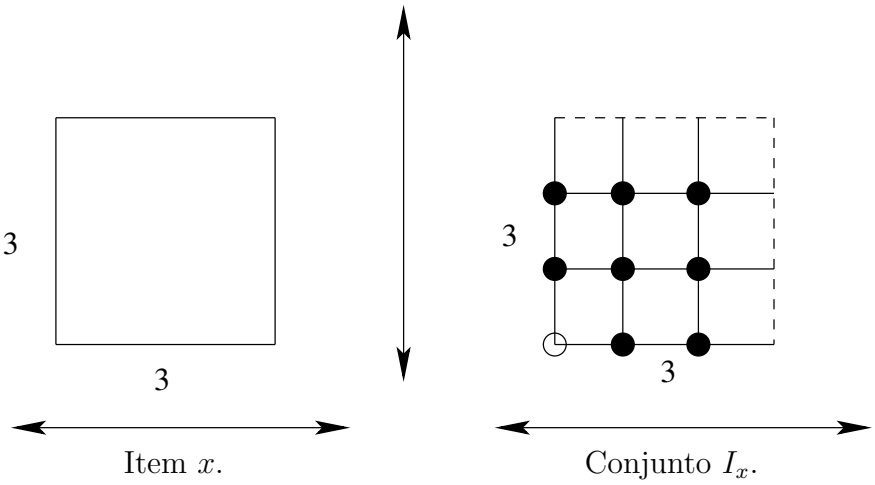


Figura 4.2: Exemplo de discretização de um item.

de itens discretizados, pode ser empacotado, tal que o empacotamento resultante tenha a menor altura possível e respeite as restrições impostas.

A figura 4.3 mostra um exemplo válido para um empacotamento de itens. Nesta figura, o item 1 de dimensões 3×3, o item 2 de dimensões 1×1 e o item 3 de dimensões 2×1 estão empacotados dentro do recipiente de dimensões 4×4.

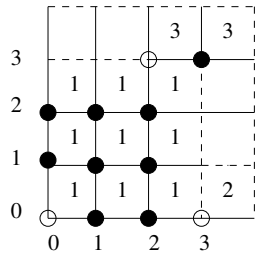


Figura 4.3: Exemplo de um empacotamento válido com itens discretizados.

Por outro lado, a figura 4.4 apresenta um exemplo de empacotamento inválido, onde temos o item 1 de dimensão 3×2, o item 2 de dimensões 2×1, o item 3 de dimensões 3×1 e o recipiente com dimensões 4×4. O empacotamento obtido nesta figura apresenta duas situações que invalida um empacotamento. A primeira é que o item 1, embora esteja apoiado sobre o item 2, não respeita as condições de equilíbrio estático de corpos rígidos, tornando este empacotamento instável. A segunda situação é que existem 2 pontos do empacotamento que fazem parte de dois itens simultaneamente. Como não deveria existir sobreposição de itens dentro do recipiente, esta interseção entre o item 1 e o item 3 também

invalida o empacotamento.

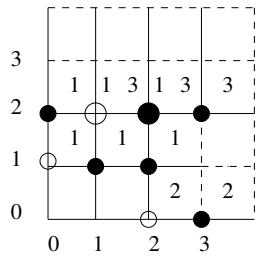


Figura 4.4: Exemplo de um empacotamento inválido com itens discretizados.

Para descobrir qual é o melhor empacotamento de itens no recipiente, associamos à cada ponto da malha um *estado de empacotamento*. Estado de empacotamento representa as características que um ponto assume na modelagem do problema. Estas características buscam mapear o problema do mundo real para um modelo matemático. Na modelagem do problema existem 4 estados.

O primeiro estado mostra onde um conjunto de pontos discretizados de um item pode ser empacotado. Desta forma identificamos a localização do item no empacotamento. O segundo estado representa o local onde este conjunto de pontos discretizados ocupará quando este item for empacotado num determinado ponto dentro do empacotamento. Com este estado definimos a área ocupada ou o lugar ocupado por um item dentro do recipiente. O terceiro estado define a ordem que cada ponto possuirá no empacotamento. Esta ordem também depende do lugar em que o item será empacotado. E por fim, temos o quarto estado que representa a faixa de altura que o item ocupa quando este é empacotado num determinado ponto do recipiente.

Na figura 4.5 temos a representação dos 4 estados quando empacotamos o item de largura 3, altura 3 e ordem 2 no ponto $(0, 0)$ de um recipiente de dimensões 4×4. Em 4.5.(a), temos o ponto que identifica a localização que o item foi empacotado no recipiente, isto é, o ponto inferior esquerdo do item. Em 4.5.(b), temos os pontos que formam a área que o item ocupa no recipiente quando este é empacotado no ponto $(0, 0)$. É importante ressaltar que a coluna 3 e linha 3 não possuem pontos, porque estes pontos são possíveis locais onde outro item pode ser empacotado. Em 4.5.(c), temos a ordem gerada no recipiente quando o item é empacotado no ponto $(0, 0)$. Em 4.5.(d), temos os pontos que fazem parte da faixa da altura quando o item é empacotado no ponto $(0, 0)$. E, em 4.5.(e) mostra todos os pontos do recipiente em que este poderia ser empacotado.

Na seção 4.2, veremos que cada um destes estados será representado por uma variável binária na formulação linear.

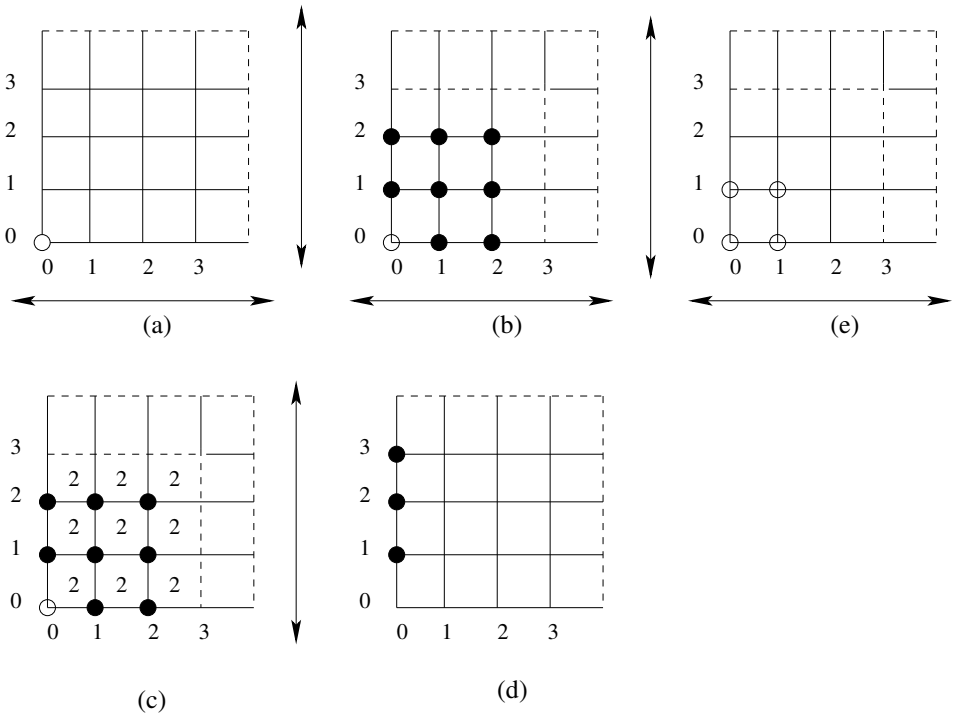


Figura 4.5: Estados de um empacotamento.

4.2 Formulações para o Problema 2EEFO

As formulações modelam o problema de empacotamento do mundo real para uma formulação matemática com objetivo de encontrar soluções boas para os problemas encontrados na prática. Apesar de temos o controle da precisão durante a geração da malha de pontos, esta formulação não resolve o problema de forma exata porque na verificação da estabilidade a força pode se localizar em um ponto fracionário não existente na malha. Por isto consideramos que ela resolve o problema de forma praticamente exata.

Como vimos na seção 4.1, a discretização do recipiente $B = (L, C)$ gera uma malha de pontos que pertencem ao conjunto $\mathcal{P}$. Seja $\mathcal{C} = \{1, 2, 3, \ldots, |\mathcal{C}|\}$ o conjunto das linhas horizontais (na direção da altura) e $\mathcal{L} = \{1, 2, 3, \ldots, |\mathcal{L}|\}$ é o conjunto das linhas verticais (na direção da largura) obtidos na discretização do recipiente. Obtemos este conjunto de linhas horizontais e verticais, de forma análoga a que obtemos o conjunto de pontos da altura e da largura de um recipiente, veja a equação (4.1).

A variável binária x_{ip} com valor 1 indica que o item i está empacotado no ponto $p \in \mathcal{P}$, isto é, com o canto inferior esquerdo sobre o ponto p, e $x_{ip} = 0$, caso contrário. Para saber as coordenadas de $p \in \mathcal{P}$ na malha de pontos, usamos o par $p = (a, b)$, sendo $a \in \mathcal{L}$,

referente ao eixo x, e $b \in \mathcal{C}$, referente ao eixo y. A variável binária $z_e = 1$ indica que existe um ponto $p_e \in \mathcal{P}$, da linha horizontal $e \in \mathcal{C}$, ocupado por um item, e $z_e = 0$, caso contrário.

Seja $\mathcal{P}_i \subseteq \mathcal{P}$ o conjunto de pontos, tal que o item i pode ser empacotado respeitando as dimensões do recipiente. Por outro lado, dado um item i empacotado no ponto $p \in \mathcal{P}$, temos o subconjunto $\mathcal{R}_{ip} \subseteq \mathcal{P}$ que contém os pontos (da malha) cobertos pelo item i, excetuando-se aqueles que estão exatamente na fronteira superior e da direita de i. Pontos cobertos são os pontos que o item i ocupa na malha quando este item é empacotado no ponto p. Dado um item i empacotado no ponto p, isto é, $x_{ip} = 1$, dizemos que um ponto $q \in \mathcal{P}$ está coberto pela região de um item i, se $q \in \mathcal{R}_{ip}$. Usamos a variável binária $y_{qi} = 1$ para indicar que o ponto $q \in \mathcal{P}$ está coberto pelo item i.

Representamos por $\mathcal{O} = \{1, 2, 3, \ldots, \mathcal{O}_{max}\}$ o conjunto que contém o valor da ordem o_i de cada item i, em que $\mathcal{O}_{max}$ é o valor da maior ordem dentro de $\mathcal{O}$. A variável binária $r_{po} = 1$ indica que o ponto $p \in \mathcal{P}$ possui ordem $o \in \mathcal{O}$, e $r_{po} = 0$, caso contrário. Observe que se $x_{ip} = 1$, então todo ponto $q \in \mathcal{R}_{ip}$ apresenta ordem exatamente o_i (do item i). Dado um ponto $p = (a, b) \in \mathcal{P}$, vamos denotar por $\lambda(p)$ o ponto $p' = (a, b')$, onde $b' = \min\{\beta \in \mathcal{C} \mid \beta > b\}$.

Quando um item $i = (l_i, c_i)$ é empacotado em p , obtemos também o conjunto $\mathcal{N}_{ip} = [b,\ b + \lceil \frac{c_i}{d_y} \rceil]$. Neste conjunto encontra-se as linhas horizontais que foram marcadas com o empacotamento de i nesta posição.

Representamos por $\mathcal{S}$ o conjunto das variáveis binárias que formam uma solução para o problema 2EEFO. $\mathcal{SI} \in \mathcal{S}$ é o conjunto de soluções instáveis. $\mathcal{X} \in \mathcal{SI}$ é o conjunto de variáveis binárias de uma solução instável. L_i é o conjunto das larguras de todos itens do problema, C_i é o conjunto das alturas e O_i é o conjunto das ordens. O domínio e as variáveis binárias apresentados serão utilizados nas duas formulações apresentadas a seguir.

A primeira formulação do problema, descrita em (4.2.1), tem como objetivo modelar um empacotamento que satisfaz as condições básicas de um empacotamento considerando a restrição de ordem de cada item dado uma malha de pontos. Já a segunda formulação, veja (4.2.2), além de fazer o que a primeira formulação faz, possui restrições para facilitar a estabilidade dos itens no empacotamento.

As condições básicas de empacotamento que ambas as formulações satisfazem garantem que não haverá sobreposição entre os itens empacotados e que um item deverá ser empacotado numa região válida dentro do recipiente. Região válida dentro do recipiente é a região que garante que o item empacotado estará dentro da área delimitada pelas

dimensões do recipiente.

As formulações inteiras descritas a seguir não tratam a estabilidade descrita no capítulo 3 devido à dificuldade em criar restrições que integrem o equacionamento proposto para a verificação da estabilidade. A verificação da estabilidade em ambas as formulações será tratada como um plano de corte no algoritmo que resolve o programa linear inteiro.

4.2.1 Primeira Formulação

A primeira formulação para o problema 2EEFO está descrita na formulação (4.3). Nesta formulação temos o recipiente $B = (L, A)$, onde L é a largura e A a altura do recipiente. Este recipiente é discretizado em uma malha de pontos cuja distância entre cada ponto no eixo x e y é, respectivamente, d_x e d_y. Além disso, temos a instância $I = (L, \infty, L_i, C_i, O_i)$ que contém os itens i (l_i, c_i, o_i), onde $i \in \mathcal{I}$, sendo $\mathcal{I}$ o conjunto de itens do problema. A quantidade de itens do problema varia de 1 até n.

A função objetivo minimiza a altura do empacotamento. Ao empacotar o item i no ponto $p = (a, b) \in \mathcal{P}$, o item i ocupará no recipiente a faixa de altura que varia de b até $b + c_i$. De acordo com a restrição (4.3.7), linhas $e \in \mathcal{C}$ correspondentes a esta faixa fazem com que a variável binária z_e seja 1 para cada uma destas linhas. Desta forma ao empacotar todos os itens de uma instância I, o somatório $\sum_{e=1}^{|\mathcal{C}|} ez_e$ fornecerá o menor valor possível para a soma das linhas horizontais ocupadas pelos itens e, consequentemente, a menor altura do empacotamento.

A restrição (4.3.1) garante que cada item i será empacotado exatamente uma vez sobre algum ponto $p \in \mathcal{P}_i$. A restrição (4.3.2) assegura que cada ponto da malha será coberto por no máximo um item, evitando, assim, que quaisquer dois itens se interceptem. A restrição (4.3.3) mostra que cada ponto da malha terá no máximo uma ordem. Note que existem pontos da malha que não serão cobertos por quaisquer itens. Isto corresponde às regiões do recipiente não ocupadas por itens.

A restrição (4.3.4) garante que, dado um item i empacotado em p, todos os pontos da região coberta por i, região $\mathcal{R}_{ip}$, terão a mesma ordem o_i. Assim para analisar a prioridade de um item no empacotamento em relação ao outro, basta comparar e validar através de algumas restrições que veremos posteriormente em (4.3.6) se os pontos deste recipiente estão respeitando o critério de que um item só pode ser empacotado em cima de outro, caso a ordem deste item seja menor ou igual à ordem do item que estiver imediatamente abaixo.

$$\min \quad \sum_{e \in \mathcal{C}} e z_e$$

sujeito a :

$$(1) \quad \sum_{p \in \mathcal{P}_i} x_{ip} = 1 \qquad \forall\ i \in \mathcal{I}.$$

$$(2) \quad \sum_{i=1}^{n} y_{pi} \leq 1 \qquad \forall\ p \in \mathcal{P}.$$

$$(3) \quad \sum_{o \in \mathcal{O}} r_{po} \leq 1 \qquad \forall\ p \in \mathcal{P}.$$

$$(4) \quad x_{ip} \leq r_{qo_i} \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i;\ \forall\ q \in \mathcal{R}_{ip}.$$

$$(5) \quad x_{ip} \leq y_{qi} \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i;\ \forall\ q \in \mathcal{R}_{ip}.$$

$$(6) \quad r_{p''o'} \leq r_{p'o''} \qquad \forall\ p' \in \mathcal{P};\ \forall\ o',\ o'' \in \mathcal{O};\ tal\ que\ o' \leq o'';\ p'' = \lambda(p').$$

$$(7) \quad x_{ip} \leq z_e \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i;\ \forall\ e \in \mathcal{N}_{ip}.$$

$$(8) \quad \sum_{\forall \mathcal{X} \in \mathcal{SI}} x_{ip} \leq n - 1 \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i;\ \forall\ q \in \mathcal{R}_{ip};\ o \in \mathcal{O};\ \mathcal{SI} \in \mathcal{S}$$

$$(9) \quad x_{ip} \in \{0, 1\} \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}.$$

$$(10) \quad y_{qi} \in \{0, 1\} \qquad \forall\ q \in \mathcal{P};\ \forall\ i \in \mathcal{I}.$$

$$(11) \quad r_{po} \in \{0, 1\} \qquad \forall\ q \in \mathcal{P};\ \forall\ o \in \mathcal{O}.$$

$$(12) \quad z_e \in \{0, 1\} \qquad e \in \mathcal{C}.$$

(4.3)

Já a restrição (4.3.5) faz com que estes pontos sejam marcados de modo que nenhum

outro item possa ser empacotado neles. Note que as restrições (4.3.2) e (4.3.5) asseguram que quaisquer dois itens nunca irão se sobrepor.

Para satisfazer a restrição de ordem imposta pelo problema, a restrição (4.3.6) garante que os itens de maior ordem sempre estarão abaixo dos itens de menor ordem. Note que esta restrição analisa a malha observando o eixo y e somente os pares de pontos que distam d_y unidades. Além disso, caso um ponto $p_i = (a, b)$ tenha ordem o_i, então o ponto imediatamente acima, $p_j = \lambda(p_i)$ (sobre a mesma reta que corta o eixo x e observando o eixo y), deve ter ordem o_j, com $o_j \leq o_i$. Já a restrição (4.3.7) garante que as linhas horizontais da malha que interceptam o item i quando este é empacotado no ponto p serão utilizadas no empacotamento.

A restrição (4.3.8) representa os cortes que serão inseridos no programa linear quando encontramos uma solução que não for estável. A verificação da estabilidade é feita pelo algoritmo (1) descrito no capítulo 3. Para realizar um corte na solução encontrada, basta inserir no programa linear uma restrição que invalida o empacotamento obtido. Garantimos a invalidação desta solução ao somar todos os pontos em que os itens foram empacotados e fazer com que este somatório seja menor do que o número de itens do empacotamento. Desta forma, o programa linear tenta encontrar outra solução na qual pelo menos um dos itens do corte inserido seja empacotado num ponto p diferente. Por fim, as restrições $(4.3.9 - 4.3.12)$ correspondem às condições de integralidade.

4.2.2 Segunda Formulação

A segunda formulação do problema está descrita na formulação 4.4. Nesta formulação temos: o recipiente $B = (L, A)$, onde L é a largura e A a altura do recipiente, a instância $I = (L, \infty, L_i, C_i, O_i)$ que contém os itens i (l_i, c_i, o_i), onde $i \in \mathcal{I}$. Além das variáveis e domínio apresentados nas seções (4.1) e (4.2), também utilizaremos o conjunto $\mathcal{F}_{ijp}$ na definição da formulação 4.2.2.

O conjunto $\mathcal{F}_{ijp}$ é o conjunto de pontos que fazem parte da interseção entre a fronteira inferior do item i com a fronteira superior do item j. Dado um item $i \in \mathcal{I}$ de dimensões (l_i, c_i), um item $j \neq i \in \mathcal{I}$ de dimensões (l_j, c_j) e um ponto $p = (p_x, p_y) \in \mathcal{P}$ definimos o conjunto $\mathcal{F}_{ijp} = \{\ \forall p' \textit{ tal que } p' = (\ a\ ,\ p_y + c_j\ ) \textit{ e } a \in [p_x - l_i + 1,\ p_x + l_j - 1] \textit{ onde } 0 \leq p_x - l_i + 1 \leq |\mathcal{L}| \textit{ e } 0 \leq p_x + l_j - 1 \leq |\mathcal{L}|\}$.

A formulação (4.4) obtém um empacotamento que favorece a estabilidade dos itens. Enquanto a formulação (4.3) preocupa-se em obter um empacotamento que satisfaz as condições inerentes ao empacotamento considerando a restrição de ordem.

A formulação (4.3) considera no seu conjunto de soluções, várias soluções que não são estáveis. Já a formulação (4.4) através da sua função objetivo além de minimizar a altura do empacotamento a ser obtido, dependendo da instância do problema, ela consegue reduzir bastante o número de soluções que não são estáveis. No momento em que o algoritmo estiver resolvendo o programa linear, serão obtidas soluções viáveis que passarão pelo corte de estabilidade quando for necessário. Desta forma, obtemos um empacotamento estável que respeita as restrições de ordem dos itens. A função objetivo da formulação (4.4) através da redução do número de soluções inviáveis visa otimizar a resolução do 2EEFO ganhando tempo na execução dos cortes.

A figura 4.6 mostra uma solução obtida ao resolver um programa linear das formulações (4.3) e (4.4) antes de aplicar o corte da estabilidade. Em 4.6.(a) temos uma solução obtida de um programa linear da formulação (4.3) e em 4.6.(b) uma solução encontrada de um programa linear da formulação (4.4). Nesta figura temos dois itens, seja g o item mais alto e b o item mais baixo. A solução ótima deste empacotamento corresponde à altura de g em ambas as formulações.

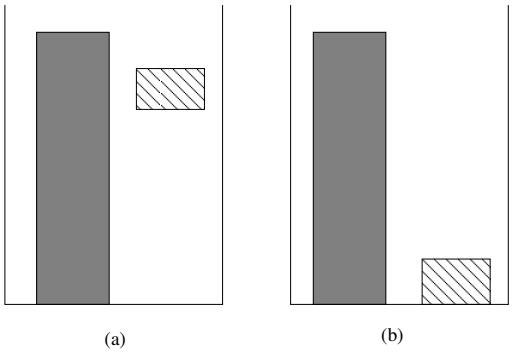


Figura 4.6: Exemplo de soluções viáveis.(a) Da formulação (4.3) e (b) Da formulação (4.4)

Perceba que na figura 4.6.(a) o item b está flutuando. Esta é uma solução ótima válida para a formulação (4.3), mas o corte que verifica a estabilidade a inviabiliza. Observe que na figura 4.6.(a), o item b pode ser empacotado em vários outros pontos, incluindo os pontos flutuantes, que a solução final será a mesma. Pontos flutuantes de um empacotamento são os pontos que não se localizam no solo e nem sobre outro item do empacotamento. Empacotamentos com pontos flutuantes são instáveis, como a formulação (4.3) não trata condições que favorecem o empacotamento, a busca para encontrar um empacotamento estável é feita através das restrições de corte.

$$\min \quad (\sum_{e \in \mathcal{C}} e z_e + \sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}_i} \varepsilon x_{ip} p_y)$$

sujeito a :

$$(1) \quad \sum_{p \in \mathcal{P}_i} x_{ip} = 1 \qquad \forall\ i \in \mathcal{I}.$$

$$(2) \quad \sum_{i=1}^{n} y_{pi} \leq 1 \qquad \forall\ p \in \mathcal{P}.$$

$$(3) \quad \sum_{o \in \mathcal{O}} r_{po} \leq 1 \qquad \forall\ p \in \mathcal{P}.$$

$$(4) \quad x_{ip} \leq r_{qo_i} \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i;\ \forall\ q \in \mathcal{R}_{ip}.$$

$$(5) \quad x_{ip} \leq y_{qi} \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i;\ \forall\ q \in \mathcal{R}_{ip}.$$

$$(6) \quad r_{p''o'} \leq r_{p'o''} \qquad \forall\ p' \in \mathcal{P};\ \forall\ o',\ o'' \in \mathcal{O};\ tal\ que\ o' \leq o'';\ p'' = \lambda(p')$$

$$(7) \quad x_{ip} \leq z_e \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i,\ \forall\ e \in \mathcal{N}_{ip}$$

$$(8) \quad x_{ip} \leq \sum_{j \neq i} x_{jp'} \qquad \forall i \in \mathcal{I};\ \forall j \neq i \in \mathcal{I};\ \forall p \in \mathcal{P}_i\ e\ \forall p' \in \mathcal{F}_{ijp}$$

$$(9) \quad \sum_{\forall \mathcal{X} \in \mathcal{SI}} x_{ip} \leq n - 1 \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}_i;\ \forall\ q \in \mathcal{R}_{ip};\ o \in \mathcal{O};\ \mathcal{SI} \in \mathcal{S}$$

$$(10) \quad z_e \in \{0, 1\} \qquad e \in \mathcal{C}.$$

$$(11) \quad x_{ip} \in \{0, 1\} \qquad \forall\ i \in \mathcal{I};\ \forall\ p \in \mathcal{P}.$$

$$(12) \quad y_{qi} \in \{0, 1\} \qquad \forall\ i \in \mathcal{I};\ \forall\ q \in \mathcal{P}.$$

$$(13) \quad r_{po} \in \{0, 1\} \qquad \forall\ o \in \mathcal{O};\ \forall\ q \in \mathcal{P}.$$

(4.4)

Na figura 4.6.(b), as soluções ótimas obtidas não permitem que o item b seja empacotado em pontos flutuantes. Isto acontece porque a função objetivo da formulação (4.4) considera um critério que favorece a estabilidade. Denominamos este critério de *ação da gravidade*, que atua de maneira semelhante a ação da gravidade exercida pelo planeta terra.

Para representar a ação da gravidade na formulação colocamos um peso ε bem pequeno para cada linha horizontal $e \in \mathcal{C}$ em que o item i pode ser empacotado. O peso atribuído ao item i empacotado na linha e será proporcional a altura desta linha. Desta forma se o item i for empacotado na linha que representa o solo, o peso atribuído a ele será nulo, se o item i for empacotado na linha 1, o peso será $1 * \varepsilon$, e para cada linha e, teremos o peso $e * \varepsilon$ adicionado ao item i. Logo, os itens empacotados estarão localizados no solo ou sobre algum item dentro do empacotamento.

A restrição (4.4.8) insere uma restrição que facilita a busca de soluções estáveis. Esta restrição garante que um item só será empacotado em regiões que favorecem a estabilidade no empacotamento. As regiões que favorecem a estabilidade são o solo e a superfície de um determinado item. Todo item empacotado no solo estará estável. Já o item i pode ser empacotado somente nos pontos em que a sua fronteira inferior tenha interseção com a fronteira superior de outro item j que já esteja empacotado e imediatamente abaixo do item i. Desta forma, esta restrição amarra os itens empacotados e ajuda a obter um empacotamento estável.

Devido à dificuldade presente na modelagem das condições de estabilidade, não foi possível inserir as restrições de estabilidade dentro das formulações (4.3) e (4.4). Como o objetivo consiste na resolução do problema 2EEFO satisfazendo a condição de estabilidade. Utilizamos um algoritmo *branch-and-cut* para resolver o programa linear e passamos o algoritmo que verifica a estabilidade como plano de corte para este algoritmo *branch-and-cut*. Desta forma encontramos a solução para o 2EEFO, que é um empacotamento estável, que respeita a restrição de ordem entre os itens e com a menor altura possível.

4.3 Algoritmos

Esta seção trará detalhes do algoritmo utilizado para resolver o problema 2EEFO. A princípio faremos uma breve introdução sobre a metodologia que usaremos na resolução do programa linear e depois retornaremos a falar do algoritmo proposto.

Na resolução do programa linear usamos como metodologia um algoritmo *branch-and-cut* que é um algoritmo *branch-and-bound* com planos de corte. Nas subseções 4.3.1 e

4.3.2 há uma breve introdução sobre o funcionamento destes algoritmos.

4.3.1 Branch-and-Bound

Branch-and-bound é um algoritmo usado para encontrar soluções de vários problemas, principalmente de otimização, consistindo de uma enumeração sistemática de soluções candidatas. O termo *branch* vem do fato que o método efetua partições no espaço das soluções e o termo *bound* refere-se ao funcionamento do algoritmo, onde um subconjunto de soluções é descartado através da utilização de limitantes superiores e inferiores.

4.3.2 Branch-and-Cut

Um algoritmo *branch-and-cut* consiste de um *branch-and-bound* em que planos de corte são gerados nos nós da árvore de busca. Ao contrário de algoritmos branch-and-bound que podem ser aplicados em diversos problemas de otimização, o branch-and-cut é um método de otimização combinatória utilizado exclusivamente para resolver problemas lineares inteiros ou inteiros mistos.

4.3.3 Algoritmo de Empacotamento

O algoritmo de empacotamento apresentado nesta seção é um algoritmo que obtém um empacotamento de itens resolvendo um programa linear inteiro. Para resolver o problema 2EEFO o algoritmo primeiramente identifica a instância I, que traz os dados dos itens e do recipiente para o problema. Posteriormente, obtém um limitante para a altura usada em um empacotamento ótimo utilizando heurísticas que serão descritas no capítulo 5 e realiza a discretização do recipiente. Em seguida, escolhe uma das formulações lineares descritas anteriormente e utiliza o conjunto de pontos discretizados para criar um programa linear inteiro P.

Este programa linear inteiro é solucionado por um resolvedor R que utiliza como metodologia um algoritmo *branch-and-cut* e que recebe o algoritmo que verifica a estabilidade para utilizá-lo na geração das desigualdades de plano de corte. Toda vez que o resolvedor obtém uma solução inteira S, realiza-se a verificação da estabilidade para esta solução. Enquanto a solução S for instável, uma restrição de corte será inserida no programa linear P e uma nova solução S deve ser obtida. Quando o algoritmo encontra uma solução S estável, armazena esta solução S e a mapea para a saída do problema. A seguir, temos uma breve descrição do algoritmo 2 e das funções utilizadas.

Algoritmo 2: Algoritmo de Empacotamento utilizando a formulação (4.4)

Entrada: Uma instância I do problema 2EEFO, o resolvedor R e o algoritmo *estavel*

Saída: Devolve um empacotamento E ou $\emptyset$.

```
2.1  E ← executaHeurísticas( I );
2.2  H ← pegaAltura( E );
2.3  M ← geraMalha( I, H );
2.4  P ← geraProgramaLinear( I, M );
2.5  S ← obtemSolucaoProgramaLinear(R, P);
2.6  enquanto S ≠ ∅ e estavel(S) for igual à instável faça
2.7      P ← insereRestricaoDeCorte(P, S);
2.8      S ← obtemSolucaoProgramaLinear(R, P);
2.9  se S = ∅ então
2.10     devolva ∅.
2.11 senão
2.12     E ← obtémEmpacotamento( S );
2.13     devolva E.
```

O algoritmo 2 recebe como entrada uma instância I do problema 2EEFO, o resolvedor R e o algoritmo *estavel* que verifica a estabilidade. Como saída, o algoritmo 2 devolve um empacotamento E que é estável ou um conjunto vazio. A instância I contém os dados dos itens e a largura do recipiente no qual deseja-se empacotar os itens, R é o resolvedor que executará o programa linear inteiro e usará o algoritmo *estavel* para realizar os cortes quando for necessário, e E é o empacotamento que contém os itens e as posições que cada um ocupa dentro do recipiente.

O algoritmo 2 cria um programa linear utilizando a formulação (4.4). Optamos por não descrever um algoritmo para a formulação (4.3) porque salvo a linha 2.4, este algoritmo é análogo ao algoritmo 2. A mudança que ocorreria na linha 2.4 é que criaríamos um programa linear com a formulação (4.3) ao invés da formulação (4.4).

A função *executaHeurísticas* recebe como entrada a instância I e executa todas as heurísticas descritas no capítulo 5. Então, devolve como solução o melhor empacotamento encontrado.

A função *pegaAltura* tem como entrada um empacotamento e devolve como saída a

altura do empacotamento, que será a altura do recipiente e também o limitante superior da altura do problema.

A função *geraMalha*, dada uma instância I e H a altura do recipiente, gera a malha de pontos discretizados dos itens e do recipiente. Como é impossível empacotar itens na altura H, a malha é gerada considerando a altura $H - 1$.

A função *geraProgramaLinear* é a função que gera um programa linear para a formulação (4.4). Esta função recebe como entrada a instância I e a malha de pontos M e tem como saída um programa linear inteiro P.

A função *obtemSolucaoProgramaLinear* tem como entrada, o resolvedor R e o programa linear P e como saída S que representa a solução ótima do programa linear.

O algoritmo *estavel* verifica a estabilidade de uma solução S e devolve como saída o valor **estavel** quando a solução S for estável e **instável**, caso contrário.

A função *insereRestricaoDeCorte* recebe o programa linear P e a solução S instável. Esta função cria a restrição de corte que inabilita que o programa linear encontre esta solução S novamente e a insere no programa linear P.

Por fim, a função *obtémEmpacotamento* tem como entrada uma solução inteira ótima que é estável S e retorna o empacotamento E. Esta função transforma a solução S, que é um vetor que contém todas as variáveis do problema já configuradas com a solução do empacotamento, num empacotamento E que contém a localização que cada item da instância I possui dentro do recipiente.

4.4 Exemplo

Abaixo segue um exemplo resolvido pelo algoritmo 2. A instância de entrada é $I = (\ 3, \infty,$ $l(1,3,2),\ c(1,1,1),\ o(1,2,3))$. Nesta instância temos 3 itens, o primeiro item tem largura 1, altura 1 e ordem 1, o segundo possui largura 3, altura 1 e ordem 2 e o terceiro e último item possui largura 2, altura 1 e ordem 3. Todas heurísticas utilizadas nesta instância encontraram altura 3 para empacotamento. Desta forma, o objetivo deste problema é empacotar os itens dentro de um recipiente de dimensões 3x3.

Neste exemplo, a malha de pontos que corresponde a discretização do recipiente está na figura 4.7. Os pontos que fazem parte da área tracejada não precisam ser inseridos na formulação, pois nenhum item pode ser empacotado na fronteira superior ou na fronteira da direita de um recipiente. Ao não analisar estes pontos de fronteiras na formulação, conseguimos diminuir o número de pontos do problema e, consequentemente, o número de variáveis e o seu tempo de execução.

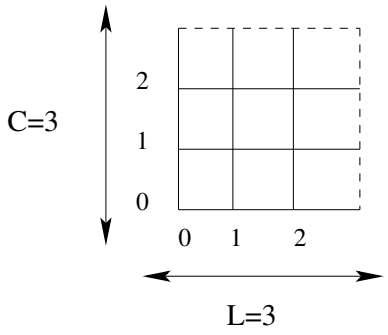


Figura 4.7: Recipiente de dimensões 3x3.

A figura 4.8 ilustra a visualização do empacotamento obtido para esta instância.

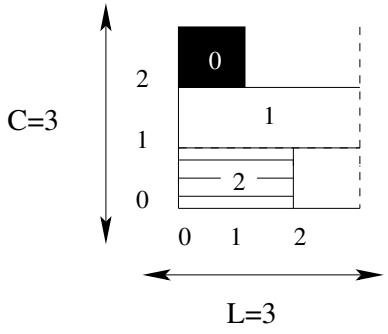


Figura 4.8: Empacotamento formado pelos itens da instância I.

A solução ótima encontrada é estável e o empacotamento obtido possui altura 3. Neste exemplo, a solução encontrada é ótima e corresponde a mesma solução encontrada pelas heurísticas. No entanto, ao analisar o tempo de execução dos algoritmos, é notório que a heurística gastou um tempo bem menor para encontrar a solução.

4.5 Considerações Finais do Capítulo

Neste capítulo mostramos como se realiza a discretização do recipiente e dos itens. Apresentamos duas formulações lineares inteiras para o problema 2EEFO e descrevemos o algoritmo usado para resolvê-los. Na discretização dos pontos mostramos como os objetos foram transformados em conjuntos de pontos. Nas formulações, explicamos as variáveis, as restrições e a função objetivo de cada uma delas. Realizamos a comparação entre as formulações e ressaltamos a importância que cada uma possui.

Capítulo 5

Heurísticas

Neste capítulo apresentaremos as heurísticas que foram desenvolvidas para encontrar um empacotamento viável e uma altura inicial que será usada pelo algoritmo de empacotamento. Todas as heurísticas deste trabalho formam e analisam empacotamento estáveis, o que dificulta ainda mais a sua elaboração.

A utilização destas heurísticas permite limitar a altura infinita do recipiente, que agora passará a ter uma altura fixa, diminuindo assim o número de pontos e de variáveis a serem analisados na resolução do problema. Ao limitar a altura do recipiente obtemos um limitante superior para a altura do empacotamento usada no algoritmo de empacotamento descrito no capítulo anterior.

Neste trabalho desenvolvemos sete heurísticas que serão descritas a seguir. Na seção 5.1 apresentaremos a *Heurística da Mochila*, na seção 5.2 a *Heurística para Recipientes*, na seção 5.3 a *Heurística de Torre*, na seção 5.4 a *Heurística Torre Mochila*, na seção 5.5 a *Heurística Torre Recipientes*, na seção 5.6 a *Heurística Pontos de Cantos* e na seção 5.7 a *Heurística Quase Exata*.

A heurística da Mochila e a heurística para Recipientes não considera a ordem dos itens para obter um empacotamento. Elas tratam o conjunto de itens que possuem a mesma altura considerando-os como um problema de empacotamento a parte. Na heurística da mochila, a largura do recipiente transforma-se na capacidade da mochila, aplicamos o algoritmo da mochila na largura dos itens que fazem parte deste conjunto e obtemos um empacotamento com itens que se encontram dentro da mochila. Se a mochila estiver cheia e ainda existir itens a serem empacotados criamos novas mochilas e resolvemos este problema até que todos os itens com a mesma altura sejam empacotados.

Já na heurística para Recipientes a largura do recipiente será o tamanho de cada recipiente unidimensional em que os itens devem ser empacotados. O problema do bin

packing unidimensional tem como entrada uma lista de itens $L = (a_1, ..., a_n)$, onde cada item tem um tamanho $s(a_i)$ e o objetivo deste problema é minimizar o número de *bins*, ou recipientes, que são necessários para empacotar todos os itens de L. Para empacotarmos os itens, atribuímos que o tamanho de cada item do problema do bin packing será correspondente a largura do item do problema e usamos a heurística *First-Fit-Decreasing (FFD)* [13] para realizar o empacotamento. Esta heurística ordena os itens em ordem decrescente de tamanho e empacota o item no primeiro recipiente que ele couber. Tanto a(s) mochila(s) como o(s) recipiente(s) obtidos serão uma faixa vertical no empacotamento obtido pela heurística da mochila e pela heurística para recipientes.

5.1 Heurística da Mochila

A heurística da Mochila obtém um empacotamento estável utilizando o algoritmo da mochila. A estratégia desta heurística consiste em formar grupos de itens que possuem a mesma altura. Cada grupo de itens possui uma altura c que corresponde a altura dos itens deste grupo e possui um número de itens que fazem parte deste grupo. Posteriormente descobrimos que o carregamento em camadas de Morabito & Arenales [35] também agrupa itens de mesma altura e emprega o algoritmo da mochila em sua solução.

Seja D o número de alturas distintas para uma instância do problema e $G = (G_1, \ldots, G_D)$ os grupos de itens com mesma altura obtidos para esta instância. Em cada um destes grupos aplicamos o algoritmo da mochila, utilizando a largura do recipiente como a capacidade da mochila. A idéia é empacotar todos os itens de um grupo G_k, onde $k \in [1, D]$, dentro de uma ou várias mochilas, formando um ou vários *blocos de itens*.

Blocos de itens é o nome dado a um conjunto de itens de um determinado grupo G_k. Cada bloco de itens terá uma largura l que será a soma das larguras de todos os itens que fazem parte deste conjunto, terá uma altura c, e terá a quantidade de itens utilizados para formar este conjunto. Desta forma, cada grupo G_k pode gerar um ou vários blocos de itens. Seja $B = (B_1, \ldots, B_b)$, onde B é o conjunto de blocos de itens gerados após aplicar a mochila em todos os grupos G_k e b é o número total de blocos de itens obtido. Ordenamos o conjunto B pela ordem decrescente de largura dos blocos e realizamos o empacotamento seguindo a ordem obtida colocando cada bloco acima do outro.

O algoritmo 3 descreve a heurística da Mochila em linhas gerais.

Algoritmo 3: Heurística da Mochila

Entrada: Um conjunto de itens I, o recipiente R e o Algoritmo da mochila.

Saída: Retorna um empacotamento E.

3.1 Seja S uma ordenação dos itens de I de forma não-crescente em c_i e para itens com mesma altura, de maneira não-crescente em l_i, onde $i \in I$.

3.2 Seja B um conjunto de blocos ordenados em forma não-crescente pela largura.

3.3 **enquanto** $S \neq \emptyset$ **faça**

3.4 numItensAlturaIgual $\leftarrow 1$;

3.5 $i \leftarrow 1$;

3.6 $j \leftarrow 1$;

3.7 Cria o grupo $G(j)$;

3.8 $G(j)_c \leftarrow S(i)_c$;

3.9 indiceInicial $\leftarrow$ i;

3.10 **se** $i \neq n$ **então**

3.11 **enquanto** $S(i)_c = S(i+1)_c$ **faça**

3.12 numItensAlturaIgual$\leftarrow$ numItensAlturaIgual + 1;

3.13 $i \leftarrow i + 1$;

3.14 $G(j)_{qtdItens} \leftarrow numItensAlturaIgual$;

3.15 indiceFinal $\leftarrow$ indiceInicial + (numItensAlturaIgual-1);

3.16 $k \leftarrow indiceInicial$;

3.17 **enquanto** $indiceInicial <= indiceFinal$ **faça**

3.18 Adiciona $S(k)$ ao grupo $G(j)$ e retira $S(k)$ de S;

3.19 $k \leftarrow k + 1$;

3.20 $i \leftarrow i + 1$;

3.21 $j \leftarrow j + 1$;

3.22 $NumeroDeGrupos \leftarrow j$;

3.23 **enquanto** *existir grupos* **faça**

3.24 $k \leftarrow 1$;

3.25 $j \leftarrow 1$;

3.26 **enquanto** $G(j)_{qtdItens} \neq 0$ **faça**

3.27 Aplica o algoritmo da mochila em $G(j)$;

3.28 Seja $B(k)$ o bloco de itens formado com os itens que estão dentro da mochila;

3.29 $G(j)_{qtdItens} \leftarrow G(j)_{qtdItens}$ - $B(k)_{qtdItens}$;

3.30 $k \leftarrow k + 1$;

3.31 $j \leftarrow j + 1$;

3.32 $NumeroDeBlocos \leftarrow k$;

3.33 **enquanto** *existir blocos* **faça**

3.34 $k \leftarrow 1$;

3.35 **se** $k = 1$ **então**

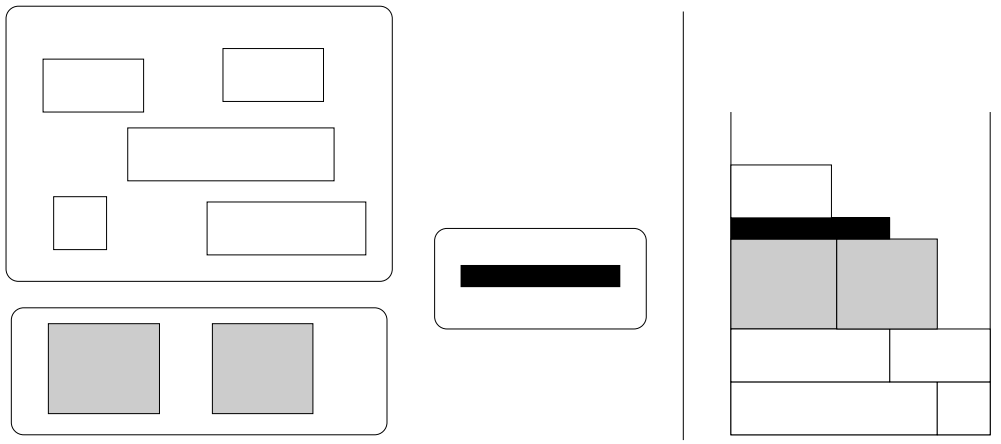
3.36 Empacota $B(k)$ em E no ponto p(0, 0);

3.37 **senão**

3.38 Empacota $B(k)$ em E no ponto p(0, $B(k)_c$);

3.39 $k \leftarrow k + 1$;

3.40 **devolva** Empacotamento E;



(a) Grupo de Itens (b) Blocos de Itens Empacotados

Figura 5.1: Empacotamento obtido pela Heurística da Mochila.

A figura 5.1 mostra um exemplo de empacotamento utilizando a heurística da mochila. Nesta figura itens que possuem a mesma altura possuem a mesma cor. Na figura 5.1.(a) temos o conjunto de itens que devem ser empacotados já agrupados nos seus respectivos grupos. Em 5.1.(b) temos um empacotamento estável formado pelos blocos de itens, cada bloco foi obtido aplicando o algoritmo da mochila nos grupos da figura 5.1.(a).

5.2 Heurística para Recipientes

A heurística para Recipientes segue a mesma metodologia da heurística da Mochila, ou seja, formamos grupos com itens que possuem a mesma altura, aplicamos um algoritmo em cada um destes grupos para formar um ou vários blocos de itens. Após analisar todos os grupos e obter todos os blocos de itens, empacotamos os blocos de forma análoga a da Heurística da Mochila.

Em cada um dos grupos aplicaremos a heurística FFD que será responsável para resolver problemas de empacotamento que envolve vários recipientes. Seja b_i o recipiente i, onde i varia de 1 até número de itens deste determinado grupo. A capacidade máxima de cada recipiente b_i será a largura do recipiente e cada recipiente b_i representará um bloco de item no final da heurística. No pior caso, cada item será empacotado dentro de um recipiente diferente.

Algoritmo 4: Heurística para Recipientes

Entrada: Um conjunto de itens I, o recipiente R e o Algoritmo FFD.

Saída: Retorna um empacotamento E.

4.1 Seja S uma ordenação dos itens de I de forma não-crescente em c_i e para itens com mesma altura, de maneira não-crescente em l_i, onde $i \in I$.

4.2 Seja B um conjunto de blocos ordenados em forma não-crescente pela largura.

4.3 **enquanto** $S \neq \emptyset$ **faça**

4.4 numItensAlturaIgual $\leftarrow 1$;

4.5 $i \leftarrow 1$;

4.6 $j \leftarrow 1$;

4.7 Cria o grupo $G(j)$;

4.8 $G(j)_c \leftarrow S(i)_c$;

4.9 indiceInicial $\leftarrow$ i;

4.10 **se** $i \neq n$ **então**

4.11 **enquanto** $S(i)_c = S(i+1)_c$ **faça**

4.12 numItensAlturaIgual$\leftarrow$ numItensAlturaIgual + 1;

4.13 $i \leftarrow i + 1$;

4.14 $G(j)_{qtdItens} \leftarrow numItensAlturaIgual$;

4.15 indiceFinal $\leftarrow$ indiceInicial + (numItensAlturaIgual-1);

4.16 $k \leftarrow indiceInicial$;

4.17 **enquanto** $indiceInicial <= indiceFinal$ **faça**

4.18 Adiciona $S(k)$ ao grupo $G(j)$ e retira $S(k)$ de S;

4.19 $k \leftarrow k + 1$;

4.20 $i \leftarrow i + 1$;

4.21 $j \leftarrow j + 1$;

4.22 $NumeroDeGrupos \leftarrow j$;

4.23 **enquanto** *existir grupos* **faça**

4.24 $j \leftarrow 1$;

4.25 Seja $B_G(j)$ o conjunto que contém todos os blocos de itens obtido do grupo de itens $G(j)$;

4.26 $B_G(j) \leftarrow$ FFD($G(j)$);

4.27 Adiciona $B_G(j)$ em B;

4.28 $j \leftarrow j + 1$;

4.29 Seja $qtdB$ o número de blocos de itens que contém no conjunto B;

4.30 $NumeroDeBlocos \leftarrow qtdB$;

4.31 **enquanto** *existir blocos* **faça**

4.32 $k \leftarrow 1$;

4.33 **se** $k = 1$ **então**

4.34 Empacota $B(k)$ em E no ponto p(0, 0);

4.35 **senão**

4.36 Empacota $B(k)$ em E no ponto p(0, $B(k)_c$);

4.37 $k \leftarrow k + 1$;

4.38 **devolva** Empacotamento E;

O algoritmo FFD recebe como entrada um conjunto G de itens que possuem a mesma altura e devolve como saída o conjunto de blocos de itens B. Este algoritmo ordena os itens de G crescentemente pela largura obtendo uma lista LI de itens ordenados. O objetivo é empacotar os itens de LI dentro do primeiro recipiente que ele couber, respeitando a ordem da lista LI e a sequência de recipientes disponíveis. Desta forma, o primeiro item de LI é empacotado dentro do primeiro recipiente disponível.

Em seguida, verifica-se o próximo item de LI caberá dentro do primeiro recipiente disponível, somando a largura deste item com a largura de todos os itens que fazem parte do recipiente verificando se o resultado obtido não excede a capacidade deste recipiente. Em caso afirmativo, o item será empacotado dentro do recipiente. Caso contrário, esta mesma análise será feita para o próximo recipiente e se repetirá até que o algoritmo consiga encontrar um recipiente para o qual este item possa ser alocado.

Após empacotar todos itens, cada recipiente que possuir pelo menos um item dentro dele será considerado um bloco de item e será inserido dentro do conjunto B. A capacidade atingida de cada recipiente corresponderá à largura deste bloco de item. Já altura do bloco será igual à altura de seus itens. O algoritmo 4 descreve a heurística para Recipientes em linhas gerais.

5.3 Heurística de Torre

A heurística de Torre obtém um empacotamento estável de itens formando uma ou várias torres de itens dentro de seu empacotamento. Temos uma torre quando um item i de dimensões (l_i, c_i, o_i) é empacotado no ponto p(a, b) e um item j de dimensões (l_j, c_j, o_j), tal que $l_j \leq l_i$, é empacotado no ponto p1(a, $b + c_i$). Para obtermos um empacotamento utilizando esta heurística adotamos a seguinte estratégia.

Primeiro ordenamos os itens em ordem não crescente de largura e obtemos a lista S que contém os itens ordenados. Seja o item $S(1)$ de dimensões (l_1, c_1) o primeiro item desta lista, selecionamos este item e o empacotamos no ponto $(0, 0)$ do recipiente.

Quando empacotamos o item $S(1)$ obtemos a primeira faixa vertical que se localizará na reta $x = l_1$. Faixa vertical é uma reta vertical que delimita as faixas de intervalos na largura do recipiente em que os itens podem ser empacotados. Cada uma destas faixas de intervalo geradas possuirá uma altura que corresponde à altura do empacotamento que se encontra dentro desta faixa. Ao empacotar o item $S(1)$ no ponto (0,0) obtemos 2 faixas de intervalo, a primeira se localiza no intervalo de largura que vai de [0,l_1) e a segunda

fica no intervalo $[l_1,L]$, onde L é largura do recipiente. Como empacotamos o item $S(1)$ na primeira faixa, neste momento a altura desta faixa será c_1, já a outra faixa, que não possui nenhum item empacotado, possuirá altura 0. A altura de cada faixa sempre será a soma das alturas dos itens que estão empacotados dentro dela.

A idéia desta heurística é de sempre tentar empacotar um item da lista S na faixa de largura que apresentar a menor altura e que caiba este item. Posteriormente descobrimos que Morabito & Arenales [35] utiliza o carreagamento em pilhas baseado na solução de problemas da mochila que possui idéia análoga a heurística de torre.

A heurística de torre pode possuir uma ou várias faixas verticais. Ela possuirá uma faixa vertical somente quando o primeiro e o segundo item a serem empacotados possuírem larguras iguais a metade da largura do recipiente. As faixas verticais são obtidas sempre que conseguimos empacotar um item no solo. Para descobrir qual será a reta correspondente a esta nova faixa vertical basta somar a largura deste último item empacotado no solo com o valor da última faixa vertical.

Após empacotarmos todos os itens, verificamos qual faixa de largura que possui a maior altura e assim encontramos a altura do empacotamento.

Esta heurística também respeita a ordem ou prioridade que cada item possui no empacotamento. Para garantir esta ordem, após obter o empacotamento, pegamos cada faixa de largura e ordenamos os itens dentro da sua respectiva faixa em ordem decrescente da ordem o dos itens. Garantimos que este empacotamento será estável empacotando cada item desta faixa de forma que o centro de gravidade de cada item desta faixa esteja exatamente acima do centro de gravidade do item que se localiza imediatamente abaixo.

A heurística de Torre pode receber como entrada tanto um conjunto de itens I ou conjunto de blocos de itens B e devolve como solução um empacotamento E. O algoritmo 5 descreve, em linhas gerais, a heurística de Torre.

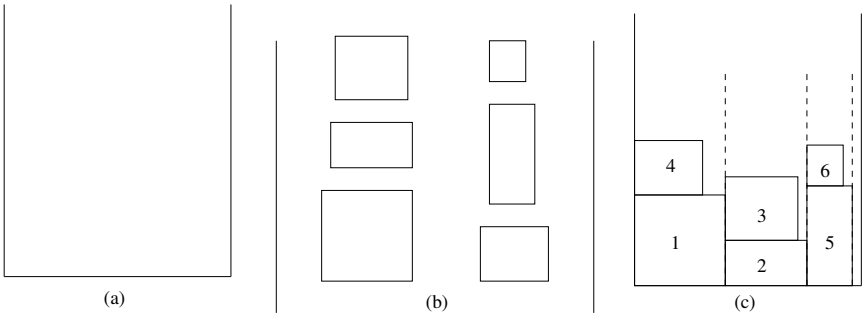


Figura 5.2: Empacotamento obtido pela Heurística de Torre

Algoritmo 5: Heurística de Torre

Entrada: Um conjunto de itens I e o recipiente R

Saída: Retorna um empacotamento E

5.1 Seja S uma ordenação dos itens de I de forma não-crescente em l_i, onde $i \in I$.
5.2 $it \leftarrow 1$; // Seja it o item atual;
5.3 Seja N um conjunto de faixas de intervalo da largura do recipiente ordenadas crescentemente em x.
5.4 $N(0)_l \leftarrow S(it)_l$;
5.5 $larguraInicialDaProximaFaixa \leftarrow 0$;
5.6 $atingiuLarguraRecipiente \leftarrow FALSE$;
5.7 **enquanto** $S \neq \emptyset$ **faça**
5.8 **se** $(larguraInicialDaProximaFaixa < R.L)$ AND $(atingiuLarguraRecipiente = FALSE)$ **então**
5.9 $k \leftarrow 0$; // k é o primeiro nível que cabe o item $S(it)$;
5.10 **enquanto** $k \leq ultimoNivel$ **faça**
5.11 $k \leftarrow k + 1$;
5.12 **se** $(atingiuLarguraRecipiente = TRUE)$ AND $(larguraInicialDaProximaFaixa + S(it).l <= R.L)$ **então**
5.13 $atingiuLarguraRecipiente \leftarrow FALSE$; $k \leftarrow 0$;
5.14 **while** $k \leq ultimoNivel$ **do**
5.15 $k \leftarrow k + 1$;
5.16 **se** $(atingiuLarguraRecipiente = TRUE)$ OR ($R.L$ *for ultrapassada*) **então**
5.17 $atingiuLarguraRecipiente \leftarrow TRUE$;
5.18 Seja j a faixa de intervalo de largura de menor altura;
5.19 empacota $S(it)$ na faixa $N(j)$ em E;
5.20 atualiza a altura de $N(j)$;
5.21 retira $S(it)$ da lista; $it \leftarrow it + 1$;
5.22 **se** $atingiuLarguraRecipiente = FALSE$ **então**
5.23 **se** $k > ultimoNivel$ **então**
5.24 $ultimoNivel \leftarrow ultimoNivel + 1$; $N(k)_l \leftarrow S(it)_l$;
5.25 $N(k)_x \leftarrow larguraInicialDoProximoNivel$;
5.26 $larguraInicialDoProximoNivel \leftarrow larguraInicialDoProximoNivel + N(k)_l$;
5.27 empacota $S(it)$ no nível $N(k)$ em E;
5.28 atualiza a altura de $N(k)$;
5.29 retira $S(it)$ da lista; $it \leftarrow it + 1$;
5.30 **devolva** Empacotamento E;

A figura 5.2 mostra um exemplo de empacotamento obtido pela heurística de Torre. Na figura 5.2.(a) temos o recipiente no qual os itens devem ser empacotados. Em 5.2.(b) temos

o conjunto de itens. E, em 5.2.(c) temos o empacotamento estável obtido. A numeração de cada item em 5.2.(c) corresponde à ordem em que estes itens foram empacotados dentro do recipiente e as retas verticais tracejadas correspondem às faixas verticais encontradas neste empacotamento.

5.4 Heurística Torre Mochila

A heurística Torre Mochila é a heurística que combina as estratégias das heurísticas 5.1 e 5.3 para obter um empacotamento. Em linhas gerais, ela utiliza a idéia da heurística da Mochila para tratar os itens que teremos de empacotar e utiliza a idéia da heurística de Torre para realizar o empacotamento.

Desta forma, dado um conjunto I de n itens, ordenamos o conjunto I em ordem decrescente pela altura de cada item e obtemos a lista $S(I)$.

Dado a lista $S(I)$, separamos os itens que possuem a mesma altura em grupos de itens. Seja $G(I)$ o conjunto de grupos de itens que possuem a mesma altura. Enquanto houver itens num determinado grupo $G(i)$, aplicamos a algoritmo da mochila em $G(i)$ e obtemos um ou vários blocos de itens para este grupo. Podemos obter vários blocos de itens para um determinado grupo $G(i)$ porque quando aplicamos o algoritmo de mochila, podem sobrar itens que não foram utilizados e neste caso aplicamos a mochila novamente. Após aplicar o algoritmo da mochila em todos os grupos do conjunto $G(I)$ e tivermos utilizados todos os itens de cada grupo, obtemos então, todos os blocos de itens necessários para o empacotamento. Para mais informações sobre os grupos e blocos de itens, vejam a seção 5.1.

Seja $B(I)$ o conjunto que contém todos os blocos de itens para uma instância I. Agora que possuímos todos os blocos de itens utilizamos a estratégia da heurística de torre para empacotá-los. Desta forma, empacotamos cada bloco de itens na faixa de largura que possuir a menor altura e sempre que possível criaremos mais faixas de larguras.

Devido ao uso da estratégia da heurística 5.1, esta heurística não considera a ordem dos itens para obter um empacotamento.

O algoritmo 6 descreve a heurística Torre Mochila.

Algoritmo 6: Heurística Torre Mochila

Entrada: Um conjunto de itens I, o recipiente R e o Algoritmo da mochila.

Saída: Retorna um empacotamento E.

```
6.1  Seja S uma ordenação dos itens de I de forma não-crescente em c_i e para itens com mesma
     altura, de maneira não-crescente em l_i, onde i ∈ I.
6.2  enquanto S ≠ ∅ faça
6.3      numItensAlturaIgual ← 1;
6.4      i ← 1;
6.5      j ← 1;
6.6      Cria o grupo G(j);
6.7      G(j)_c ← S(i)_c;
6.8      indInicial ← i;
6.9      se i ≠ n então
6.10         enquanto S(i)_c = S(i+1)_c faça
6.11             numItensAlturaIgual← numItensAlturaIgual + 1;
6.12             i ← i + 1;
6.13     G(j)_qtdItens ← numItensAlturaIgual;
6.14     indFinal ← indInicial + (numItensAlturaIgual-1);
6.15     k ← indInicial;
6.16     enquanto indInicial <= indFinal faça
6.17         Adiciona S(k) ao grupo G(j) e retira S(k) de S;
6.18         k ← k + 1;
6.19     i ← i + 1;
6.20     j ← j + 1;
6.21 NumeroDeGrupos ← j;
6.22 enquanto existir grupos faça
6.23     k ← 1;
6.24     j ← 1;
6.25     enquanto G(j)_qtdItens ≠ 0 faça
6.26         Aplica o algoritmo da mochila em G(j);
6.27         Seja B(k) o bloco de itens formado com os itens da mochila;
6.28         G(j)_qtdItens ← G(j)_qtdItens - B(k)_qtdItens;
6.29         k ← k + 1;
6.30     j ← j + 1;
6.31 NumeroDeBlocos ← k;
6.32 Aplica o algoritmo 5 no conjunto de blocos de itens B e obtemos o empacotamento E;
6.33 E ← HeuristicaDeTorre(B, R);
6.34 devolva Empacotamento E;
```

5.5 Heurística Torre Recipientes

A heurística Torre Recipientes assim como a heurística torre mochila combina a estratégia de duas heurísticas para obter um empacotamento. Ela conserva a forma com que a heurística Torre Mochila realiza o seu empacotamento e muda somente o modo como os itens deverão ser empacotados. Para realizar o empacotamento utilizamos a mesma idéia da heurística de torre e para o tratamento dos itens usamos a idéia da heurística para Recipientes.

Desta forma, dado um conjunto I de n itens, ordenamos o conjunto I em ordem decrescente pela altura de cada item e obtemos a lista $S(I)$. Dado a lista $S(I)$, separamos os itens que possuem a mesma altura em grupos de itens. Seja $G(I)$ o conjunto de grupos de itens com a mesma altura. Aplicamos a algoritmo FFD em $G(i)$ e obtemos um ou vários blocos de itens para este grupo. Após aplicar o algoritmo FFD em todos os grupos do conjunto $G(I)$, obtemos todos os blocos de itens necessários para o empacotamento.

Seja $B(I)$ o conjunto que contém todos os blocos de itens para uma instância I. Com todos os blocos de itens definidos, utilizamos a estratégia da heurística de torre para empacotá-los.

Esta heurística não analisa a ordem dos itens para obter um empacotamento. O algoritmo 7 descreve, em linhas gerais, a heurística Torre Recipientes.

5.6 Heurística Pontos de Canto

A heurística Pontos de Canto busca empacotar os itens nas posições mais baixas e a esquerda do recipiente. Para seguir esta ordem empacotamos os itens respeitando a ordem dos pontos na lista de pontos de canto. Os itens também serão colocados no recipiente seguindo uma ordem imposta.

Pontos de canto são os pontos obtidos pela combinação das dimensões dos itens empacotados. Dado um empacotamento, os pontos de canto são os pontos extremos de um empacotamento. Para obter os pontos de canto pegamos o item mais alto do empacotamento, percorremos a fronteira superior e a fronteira da direita deste item até encontrarmos outro item. Quando encontramos outro item, repetimos este procedimento até chegar ao solo. As extremidades que encontramos neste caminho (sentido para baixo e para direita) são os pontos de cantos deste empacotamento.

Algoritmo 7: Heurística Torre Recipientes

Entrada: Um conjunto de itens I, o recipiente R e o Algoritmo FFD.

Saída: Retorna um empacotamento E.

7.1 Seja S uma ordenação dos itens de I de forma não-crescente em c_i e para itens com mesma altura, de maneira não-crescente em l_i, onde $i \in I$.
7.2 **enquanto** $S \neq \emptyset$ **faça**
7.3 numItensAlturaIgual $\leftarrow 1$;
7.4 $i \leftarrow 1$;
7.5 $j \leftarrow 1$;
7.6 Cria o grupo $G(j)$;
7.7 $G(j)_c \leftarrow S(i)_c$;
7.8 indInicial $\leftarrow$ i;
7.9 **se** $i \neq n$ **então**
7.10 **enquanto** $S(i)_c = S(i+1)_c$ **faça**
7.11 numItensAlturaIgual$\leftarrow$ numItensAlturaIgual + 1;
7.12 $i \leftarrow i + 1$;
7.13 $G(j)_{qtdItens} \leftarrow numItensAlturaIgual$;
7.14 indFinal $\leftarrow$ indInicial + (numItensAlturaIgual-1);
7.15 $k \leftarrow indInicial$;
7.16 **enquanto** $indInicial <= indFinal$ **faça**
7.17 Adiciona $S(k)$ ao grupo $G(j)$ e retira $S(k)$ de S;
7.18 $k \leftarrow k + 1$;
7.19 $i \leftarrow i + 1$;
7.20 $j \leftarrow j + 1$;
7.21 $NumeroDeGrupos \leftarrow j$;
7.22 **enquanto** *existir grupos* **faça**
7.23 $j \leftarrow 1$;
7.24 Seja $B_G(j)$ o conjunto que contém todos os blocos de itens obtido do grupo de itens $G(j)$;
7.25 $B_G(j) \leftarrow$ FFD($G(j)$);
7.26 Adiciona $B_G(j)$ em B;
7.27 $j \leftarrow j + 1$;
7.28 Aplica o algoritmo 5 no conjunto de blocos de itens B e obtemos o empacotamento E;
7.29 $E \leftarrow HeuristicaDeTorre(B, R)$;
7.30 **devolva** Empacotamento E;

Seja o ponto de canto p de coordenadas (x, y) e i o item de dimensões (l_i, c_i). Ao empacotar i em p, obtemos os pontos de canto $(x, y + c_i)$ e $(x + l_i, c_i)$, que são pontos extremos de i neste empacotamento. Dependendo da configuração do empacotamento podemos obter outros pontos de canto além dos citados para o item i. Abaixo a figura

5.3 mostra alguns exemplos de pontos de canto.

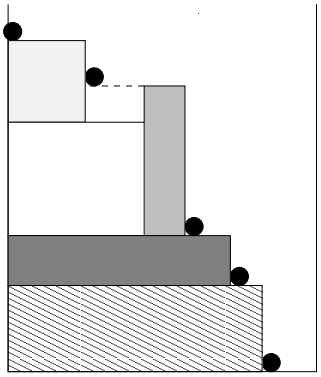


Figura 5.3: Exemplos de Pontos de Canto dentro de um empacotamento.

Seja LPC a lista de pontos de canto obtido de um empacotamento. Nesta lista estarão todos os pontos de canto obtidos que ainda não foram utilizados no empacotamento. Os pontos desta lista estarão ordenados em ordem crescente no eixo y, e em caso de empate, em ordem crescente no eixo x. Desta forma, tentaremos sempre empacotar os itens nas posições mais baixas e a esquerda do recipiente.

Seja LI a lista que contém os itens que serão empacotados. Os itens desta lista também respeitam uma ordem de entrada no empacotamento. Esta ordem de entrada define qual será a sequência de itens a empacotar dado um critério de ordenação. Os critérios de ordenação para a lista LI são:

1. Ordenar os itens pela largura e altura;
2. Ordenar os itens pela ordem e largura;
3. Ordenar os itens pela área e largura;

Todos os critérios de ordenação citados tem a sua ordenação principal dada pelo primeiro nome e em caso de empate, desempata pelo segundo nome. Quando utilizamos o primeiro critério de ordenação, primeiro estamos ordenando os itens em ordem decrescente de largura e, em caso de empate, ordenamos os itens empatados em ordem crescente de altura. O segundo critério, ordena os itens em ordem decrescente da ordem de cada item e, em caso de empate, desempatamos ordenando os itens em ordem decrescente de largura. O terceiro e último critério, ordena os itens por ordem decrescente da área de cada item e como critério de desempate ordena os itens em ordem decrescente de largura.

Com o critério de seleção escolhido, definimos a sequência em que os itens devem ser empacotados.

Para descobrir em qual ponto de canto o item i da lista LI será empacotado simulamos que i será empacotado no ponto de canto pc da lista LPC. Se nesta simulação o item i respeitar as condições básicas de empacotamento, empacotamos i no ponto pc, atualizamos a lista de pontos canto, utilizamos o próximo item da lista e tentamos empacotá-lo em algum ponto de canto. Agora, caso o item i não possa ser empacotado no ponto de canto pc, escolhemos outro ponto de canto e tentamos empacotar o item i nele.

As condições básicas de empacotamento que são analisadas e verificadas no momento em que tentamos empacotar o item no ponto de canto são a sobreposição de itens e estabilidade do empacotamento quando o item é adicionado neste ponto. Na sobreposição de itens, verificamos se o item está localizado dentro do recipiente e observamos se existem itens que ocupam a mesma área ou região neste empacotamento. Já na estabilidade, pegamos o empacotamento de itens existente e através do algoritmo descrito em 1 verificamos se ao adicionar o item i no ponto pc obtemos um novo empacotamento que respeite as condições de equilíbrio dos corpos rígidos. O algoritmo 8 descreve a heurística pontos de cantos em linha gerais.

Algoritmo 8: Heurística Pontos de Canto

Entrada: Lista de itens ordenados LI, uma Lista de pontos de Cantos PC ordenados e R o recipiente que receberá os itens.

Saída: Retorna um empacotamento E.

```
8.1  quantidadePontos ← 0;
8.2  para cada i ∈ LI faça
8.3      para cada p ∈ PC faça
8.4          se (verificaRecipiente(i,p,R) = True) AND
             (verificaSobreposicao(i,p,E) = True) AND (estavel(i,p,E) = True) então
8.5              Empacota i no ponto p em E;
8.6              Retira i de LI;
8.7              PC ← geraPontosDeCanto(E);
8.8              quantidadePontos ← 0;
8.9          senão
8.10             quantidadePontos ← quantidadePontos +1;
8.11     se (quantidadePontos = quantidadePontosCantos) então
8.12         Coloca i na última posição de LI;
8.13 devolva Empacotamento E;
```

Neste algoritmo a função *verificaRecipiente(i,p,R)* recebe um item i, um ponto p e um recipiente R, e verifica se ao empacotar o item i no ponto p, se o item i esta localizado dentro do recipiente R.

A função *verificaSobreposicao(i,p,E)* recebe um item i, um ponto p e um empacotamento E e verifica se ao empacotar o item i no ponto p, se o item i sobrepõe algum item que esteja empacotado em E. Já a função *estavel(i,p,E)* recebe um item i, um ponto p e um empacotamento E como entrada e verifica se ao empacotar o item i no ponto p, se o empacotamento E continua estável. Por fim, temos a função *geraPontosDeCanto(E)* que recebe um empacotamento E como entrada e devolve como saída uma lista de pontos de cantos.

5.7 Heurística Quase Exata

A heurística Quase Exata obtém o empacotamento estável transformando o conjunto de itens e o recipiente do problema num programa linear que será resolvido por um resolvedor. Esta heurística se assemelha bastante ao algoritmo de empacotamento descrito na subseção 4.3.3.

A altura do recipiente assim como no algoritmo de empacotamento é infinita. Desta forma, no início do problema utilizamos as heurísticas descritas anteriormente para definir uma altura inicial boa para o recipiente.

A heurística quase exata possui esta denominação, porque assim como o algoritmo de empacotamento, ela recebe e trata os dados de entrada transformando-os num programa linear inteiro de acordo com a formulação escolhida e assim resolve o problema. O conjunto de larguras utilizado por esta heurística na representação dos dados das larguras no programa linear.para uma dada instância é normalmente menor do que o conjunto de largura do algoritmo de empacotamento. Esta diminuição acontece porque esta heurística seleciona as larguras que serão usadas através da combinação linear das larguras dos itens de uma determinada instância. Seja LL o conjunto formado pelas possíveis larguras, i um item que possui altura c_i e largura l_i, n a quantidade de itens que devem ser empacotado e $\alpha \in \{0, 1\}$. A equação (5.1) mostra como é o obtido o conjunto LL

$$LL \ = \ \sum_{i=1}^{n} \alpha_i \times l_i. \tag{5.1}$$

Após a obtenção do conjunto LL restringimos o conjunto de pontos a serem estudados no problema e assim obtemos uma resposta mais rápida. No entanto, excluímos possíveis pontos em que itens poderiam ser empacotados, como observaremos no exemplo a seguir.

Neste exemplo queremos empacotar 2 itens, o item A de largura 2 e altura 3 e o item B de largura 3 e altura 2, num recipiente de dimensões 6×6. No algoritmo de

empacotamento, o recipiente será discretizado e obtemos o conjunto de larguras $X = \{0, 1, 2, 3, 4, 5\}$ e de alturas $Y = \{0, 2, 3, 5\}$, como mostra a equação (4.1). Os itens poderão ser empacotados em qualquer ponto p pertencente ao conjunto dos pontos da malha. Utilizando a heurística quase exata, trabalharemos com a ordem crescente das larguras obtidas na combinação linear das larguras dos itens. Neste exemplo, o conjunto LL será igual a $LL = \{0, 2, 3, 5\}$.

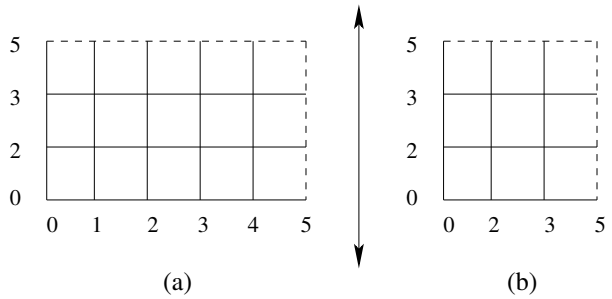


Figura 5.4: Exemplos de discretização do recipiente para o Algoritmo de Empacotamento e para a Heurística Quase Exata.

A figura 5.4 mostra como fica a discretização do recipiente para o algoritmo de empacotamento e para a heurística quase exata. Em 5.4.(a) temos o recipiente discretizado utilizando o algoritmo de empacotamento e 5.4.(b) o recipiente discretizado obtido pela heurística quase exata.

Note que há uma diminuição no conjunto de pontos obtido pela a heurística ao compará-lo com o conjunto LL. Como o programa linear possui o mesmo comportamento para a heurística e para o algoritmo de empacotamento, o fato de possuirmos menos pontos, resultará em menos variáveis para o programa e, consequentemente, poderemos obter uma solução mais rápida. O lado negativo desta heurística é que os pontos que foram descartados podem ser importantes para algumas instâncias, pois dependendo do problema, ao utilizar os pontos que foram descartados poderíamos obter um empacotamento estável com uma altura menor, por exemplo.

No algoritmo de empacotamento, a altura do recipiente também é obtida através da combinação linear das alturas dos itens. No entanto, limitamos o problema com a altura obtida nas heurísticas, não precisando assim utilizar todos os pontos obtidos na combinação linear. Para a altura não há problemas em trabalhar com a combinação linear pois os itens sempre serão empacotados respeitando esta ordem. Já na largura isto não acontece porque ao deslocarmos algum item do empacotamento de uma unidade obteremos uma nova combinação linear que não corresponderá a que está sendo utilizada

no problema.

Excetuando a parte do tratamento que é dado nos pontos obtidos na "discretização"da largura do recipiente, o resto da heurística apresenta o mesmo comportamento que o algoritmo de empacotamento. Para mais informações veja o capítulo 4.

As funções novas com relação ao algoritmo 2 são as funções *geraLargura* e *geraMalha2*. A função geraLargura recebe como entrada uma instância I e devolve W que é o conjunto de larguras do recipiente que serão utilizadas no empacotamento. Este conjunto é obtido através da combinação linear das larguras dos itens. Já a função geraMalha2 dada uma instância I, a altura do recipiente H e o conjunto de larguras do recipiente W, gera a malha ou conjunto de pontos discretizados dos itens e do recipiente que serão usados no programa linear. O algoritmo 9 descreve a Heurística Quase Exata.

Algoritmo 9: Heurística Quase Exata

Entrada: Uma instância I do problema 2EEFO, o resolvedor R e o algoritmo *estavel*

Saída: Devolve um empacotamento E ou $\emptyset$.

```
9.1  E ← executaHeurísticas( I );
9.2  H ← pegaAltura( E );
9.3  W ← geraLargura( I );
9.4  M ← geraMalha2( I, H, W );
9.5  P ← geraProgramaLinear( I, M );
9.6  S ← obtemSolucaoProgramaLinear(R, P);
9.7  enquanto S ≠ ∅ e estavel(S) for igual à instável faça
9.8      P ← insereRestricaoDeCorte(P, S);
9.9      S ← obtemSolucaoProgramaLinear(R, P);
9.10 se S = ∅ então
9.11     devolva ∅;
9.12 senão
9.13     E ← obtémEmpacotamento( S );
9.14     devolva E;
```

5.8 Execução das Heurísticas

Cada uma das heurísticas anteriores pode retornar valores distintos para uma mesma instância do problema. Isto é justificado pela metodologia e comportamento que cada uma possui. Nesta seção, demonstraremos como combinamos as heurísticas citadas para achar a melhor solução para uma dada instância I.

Aplicamos I na heurística da Mochila, na heurística para Recipientes, na heurística de Torre, na heurística Torre Mochila, na heurística Torre Recipientes e na heurística de Pontos de Cantos. Todas as heurísticas retornam um empacotamento estável. Comparamos a altura de cada empacotamento e escolhemos a altura do menor empacotamento. Seja $hMIN$ a altura do menor empacotamento e $hMIN'$ a altura correspondente ao próximo ponto de discretização logo abaixo de $hMIN$.

Em seguida, aplicamos a instância I na heurística Quase Exata e passamos como altura do recipiente a altura $hMIN'$ e executamos a heurística. Caso esta heurística encontre uma solução para a instância I, certamente esta solução possuirá um empacotamento melhor que as outras e atualizamos o valor de $hMIM$ com a altura deste empacotamento. Caso contrário, como a heurística não conseguiu encontrar uma solução para I, podemos afirmar que a altura encontrada nas heurísticas anteriores é a melhor e assim conservamos o valor de $hMIM$.

Passamos para a heurística Quase Exata o valor de $hMIM'$ porque já encontramos um empacotamento que tenha a altura $hMIM$ nas heurísticas anteriores. Desta forma esta heurística que é mais pesada, executará de forma otimizada, pois menos pontos precisarão ser analisados no programa linear.

5.9 Considerações Finais do Capítulo

As heurísticas apresentadas retornam soluções estáveis para o nosso problema. As heurísticas 5.1, 5.2, 5.3, 5.4, 5.5 e 5.6 apresentam soluções num tempo bem pequeno. Já a heurística 5.7 apresenta a solução que se não for à ótima certamente estará muito próxima dela. Porém esta heurística pelo fato de resolver um programa linear inteiro tem o ônus no seu tempo de execução. Na execução das heurísticas combinamos todas heurísticas descritas e retornamos a melhor altura encontradas.

Capítulo 6

Resultados Computacionais

Este capítulo apresenta a análise dos resultados computacionais obtidos pelo algoritmo de empacotamento e pelas heurísticas.

O algoritmo proposto foi codificado na linguagem C/C++ e os experimentos ocorreram em um computador com processador Intel® Core^{TM} 2 Quad 2, 4 GHz, memória RAM de 4 GB e sistema operacional Linux. O modelo de programação inteira foi resolvido usando a versão 19.0 da ferramenta Xpress-Optimizer. Também, limitamos o tempo de CPU em 4 horas durante a resolução de cada instância.

É notável que nosso modelo de programação inteira é inadequado para resolver instâncias de grande porte. Note que há um elevado número de variáveis, as quais aumentam à medida que refina-se a malha de pontos que compõe o recipiente.

Em frente a esta dificuldade, os testes realizados contemplam algumas instâncias adaptadas da OR-Library [3]. Como não há instâncias específicas para o problema em questão, muito menos resultados para serem comparados, também geramos um conjunto aleatório de instâncias e apresentamos os resultados obtidos.

6.1 Resultados relacionados à estabilidade

As instâncias da OR-Library correspondem às instâncias $ngcut01 - ngcut11$ e $cgcut01 - cgcut03$ usadas pelo problema da Mochila Bidimensional Limitada em que os itens possuem orientação fixa e os cortes devem ser não-guilhotinados e guilhotinados, respectivamente. A solução ótima destas instâncias podem ser encontradas nos trabalhos de [7, 11, 23].

Nas instâncias $ngcut01 - ngcut11$ e $cgcut01 - cgcut03$ desprezamos a altura do reci-

piente e geramos randomicamente um valor de ordem para cada item.

Por outro lado, as instâncias geradas randomicamente estão divididas em três grupos, cada qual com cinco instâncias. A largura do recipiente de cada grupo é respectivamente 10, 15, 20. Além disso, para cada grupo de instâncias, o número total de itens varia de 5 a 13, observando de forma crescente somente os números ímpares neste intervalo. As dimensões de cada item variam de 10% a 90% da largura do recipiente e cada item possui um valor de ordem gerado no intervalo $[1, 13]$. Chamaremos tais instâncias de $ef11, \ldots, ef15$, para o grupo 1; $ef21, \ldots, ef25$, para o grupo 2; e, $ef31, \ldots, ef35$, para o grupo 3.

Durante a geração da malha de pontos foi necessário estabelecer uma altura inicial, C, para o recipiente. Este valor corresponde à melhor altura encontrada ao aplicar as heurísticas nesta instância.

A tabela 6.2 apresenta os resultados para instâncias que utilizam a formulação descrita na seção 4.2.1. Nesta formulação, o problema 2EEFO é resolvido sem utilizar condições que favorecem os critérios de estabilidade dentro da própria formulação. Já a tabela 6.3 utiliza a formulação 4.2.2 que corrabora com a obtenção de um empacotamento estável.

Em cada linha das tabelas 6.2 e 6.3 temos as informações: nome da instância; dimensões do recipiente; quantidade de itens; altura da faixa; porcentagem da área da faixa utilizada; tempo (de CPU) gasto em segundos; número total de nós explorados; número total de cortes de estabilidade adicionados.

Apesar da metodologia destes resultados se diferenciarem quanto ao favorecimento ou não da estabilidade, obtemos a mesma altura de empacotamento na execução de todas as instâncias.

As instâncias que possuem $(*)$ ao lado do seu nome são as instâncias que encontraram a melhor solução até aquele momento de execução.

Na tabela 6.1 temos o gap de otimalidade encontrado para cada uma das instâncias $(*)$.

Seja S a solução obtida e S^* a solução ótima, a equação 6.1 mostra como é feito o cálculo do gap de otimalidade.

$$gap = \frac{|S - S^*|}{S^*}. \tag{6.1}$$

Instâncias como $ngcut5$, $ngcut8$, e $ef24$ que não possuem solução na tabela 6.2, possuem na tabela 6.3.

A maioria das instâncias da tabela 6.3 tiveram os resultados encontrados em um tempo menor do que as instâncias da tabela 6.2. De modo geral, os resultados da tabela

6.3 apresentam menos corte do que a tabela 6.2.

Como as instâncias das duas tabelas são iguais podemos creditar estas melhorias às restrições que favorecem a estabilidade. Com a utilização destas restrições, as soluções encontradas tendem a ser mais estáveis do que as da tabela 6.2 possibilitando a diminuição do número de cortes e no tempo de execução para a maioria das instâncias.

Tabela 6.1: Gap de otimalidade.

Tabela	Instância	Gap (%)
Tabela 6.2	cgcut01	0.02
Tabela 6.2	ngcut02	0.16
Tabela 6.2	ef13	0.23
Tabela 6.2	ef25	0.45
Tabela 6.2	ef31	0.09
Tabela 6.3	ngcut05	0.39
Tabela 6.3	ngcut08	0.43
Tabela 6.3	ef24	0.28
Tabela 6.3	ef25	0.39

Ambas as abordagens consideradas nas tabelas 6.2 e 6.3 resolvem o problema em questão para os casos em que o empacotamento considera a restrição de ordem e para os casos que não consideram. Não apresentamos resultados para comparar empacotamentos quanto à ordem porque consideramos que o problema de empacotar estes itens é distinto do problema que não possui ordem.

Observando as tabelas 6.2 e 6.3, vimos que o algoritmo não conseguiu retornar a solução de algumas instâncias dentro do tempo máximo de CPU adotado. Em outras palavras, após quatro horas de processamento, não conseguimos solução ótima estável para as instâncias cujas linhas/colunas estão marcadas com $-$, ou seja, para as instâncias $cgcut02$, $cgcut03$, $ngcut06$, $ngcut09 - ngcut11$, $ef33 - ef35$.

Note que apesar de serem instâncias pequenas, com poucos itens, o algoritmo gastou um satisfatório tempo para resolvê-las, incluindo a geração de muitos nós e de chamadas a rotina de corte. Como mencionado, a rotina de estabilidade consome tempo $O(n^4)$, sendo n o número total de itens para uma instância de entrada.

Através disto, temos que nossa estratégia é eficiente para tratar instâncias de pequeno até médio porte.

6.2 Resultado obtido pelas Heurísticas

Esta seção apresenta os resultados obtidos para algumas instâncias utilizando apenas heurísticas. Será apresentado os resultados para as instâncias mostradas na seção 6.1 e para instâncias criadas randomicamente. As instâncias criadas randomicamente tratam os casos em que se deve empacotar uma quantidade maior de itens dentro de um recipiente e os casos em que os itens a serem empacotados apresentam dimensões superiores a 20. As instâncias geradas randomicamente estão divididas em duas categorias.

Na primeira categoria, temos as instâncias cujo recipiente possui largura 50. Os itens a serem empacotados estão divididos em três grupos, cada qual possui uma quantidade de itens diferentes. O primeiro grupo possui 10 itens, o segundo 15 e o terceiro 20. Na apresentação dos resultados o nome das instâncias serão distinguidas pelo número de itens de cada grupo. Cada item gerado possui uma dimensão máxima de 70% da largura do recipiente e uma dimensão mínima de 40% da largura do recipiente.

Na segunda categoria, as instâncias também estão divididas em três grupos de forma semelhante aos da primeira categoria. Desta forma, o número de itens de cada grupo será 10, 15, 20, respectivamente. Porém em cada grupo da segunda categoria os itens gerados possuem no máximo três opções de altura. Criamos estas instâncias para verificar o desempenho das heurísticas para as situações em que os itens a serem empacotados possuem um ou vários conjuntos de itens com a mesma altura.

Para cada grupo de uma categoria temos 10 instâncias. Chamaremos tais instâncias de $10/ins1, \ldots, 10/ins10$, para o grupo 1; $15/ins1, \ldots, 15/ins10$, para o grupo 2; e, $20/ins1, \ldots, 20/ins10$, para o grupo 3;

Em cada linha das tabelas 6.4, 6.6 e 6.7 temos as informações: nome da instância; quantidade de itens; dimensões do recipiente; altura da heurística da mochila; altura da heurística de torre; altura da heurística da torre mochila; altura da heurística pontos de cantos; altura da heurística para recipientes; altura da heurística torre recipientes.

Já para cada linha da tabela 6.5 temos informações: nome da instância; quantidade de itens; dimensões do recipiente; altura da heurística quase exata para as instâncias que não possuem restrições que apóiam a estabilidade (hqeSemApoio); tempo da heurística quase exata para as instâncias que não possuem restrições que apóiam a estabilidade (hqeSemApoio); altura da heurística quase exata para as instâncias que possuem restrições que apóiam a estabilidade (hqeComApoio); tempo da heurística quase exata para as instâncias que possuem restrições que apóiam a estabilidade (hqeComApoio);

A tabela 6.4 apresenta o resultado para as instâncias analisadas na seção 6.1. Todas as

instâncias da tabela 6.4 foram resolvidas em menos de 1 segundo. Como esperávamos, a relação entre heurísticas e um algoritmo praticamente exato é estreita quanto ao tempo e à precisão. Na heurística se ganha no tempo e para grande maioria das instâncias perde-se em precisão. Já o algoritmo praticamente exato ganha em precisão e perde no tempo.

Ao analisar os resultados das tabelas 6.2, 6.3 com a tabela 6.4, percebe-se que os resultados obtidos pelas heurísticas foram bons e estão bem próximos dos que foram obtidos pelo algoritmo praticamente exato.

A tabela 6.5 compara a execução da heurística quase exata para os casos em que as restrições apóiam e não apóiam a estabilidade. Comparando o resultado desta heurística com o resultado do algoritmo praticamente exato nota-se que a maioria das instâncias executadas pela a heurística apresentam a mesma resposta encontrada no algoritmo praticamente exato utilizando um tempo menor.

Comparando o desempenho só das heurísticas para as instâncias apresentadas na seção 6.1 apesar de todas obterem um rendimento semelhante, a heurística de torre teve um ligeiro destaque em relação às demais. Ao analisar a tabela 6.6 temos que as heurísticas apresentam um desempenho parecido, porém para algumas instâncias as heurísticas torre mochila, pontos de cantos, para recipientes e torre recipientes apresentam resultados mais satisfatórios. Quando observamos a tabela 6.7, na qual os itens possuem no máximo 3 opções de altura, fica perceptível que as heurísticas que tratam os itens com alturas iguais, separando-os em grupos distintos, apresentam os melhores resultados.

6.3 Considerações Finais do Capítulo

Comparamos o algoritmo praticamente exato quanto as suas restrições que favorecem ou não a estabilidade e vimos que as que favorecem apresentam resultados melhores. Foi possível relacionar o desempenho do algoritmo praticamente exato com os das heurísticas. Descobrimos que as heurísticas apresentam desempenhos similares para a grande maioria das instâncias, mas quando tratamos instâncias que tem grupos de itens com a mesma altura algumas heurísticas se destacam mais.

Tabela 6.2: Resultados para as instâncias *ngcuts*, *cgcuts* e $ef's$ sem as restrições que apóiam a estabilidade.

Instância	$B=(L,C)$	N. de Itens	Altura da Faixa	Área (%)	Tempo (s)	N. de Nós Explorados	N. de Cortes
cgcut01*	(15, ∞)	7	8	0.82	15002	3691816	1
cgcut02	(40, ∞)	10	–	–	–	–	–
cgcut03	(40, ∞)	20	–	–	–	–	–
ngcut01	(10, ∞)	5	11	0.86	2	62	2
ngcut02*	(10, ∞)	7	11	0.61	15000	1966423	0
ngcut03	(10, ∞)	10	–	–	–	–	–
ngcut04	(15, ∞)	5	8	0.84	361	14501	0
ngcut05	(15, ∞)	7	–	–	–	–	–
ngcut06	(15, ∞)	10	–	–	–	–	–
ngcut07	(20, ∞)	5	9	0.81	24	215	2
ngcut08	(20, ∞)	7	–	–	–	–	–
ngcut09	(20, ∞)	10	–	–	–	–	–
ngcut10	(30, ∞)	5	–	–	–	–	–
ngcut11	(30, ∞)	7	–	–	–	–	–
ef11	(10, ∞)	5	8	0.95	17	5547	0
ef12	(10, ∞)	7	7	0.93	33	1923	0
ef13*	(10, ∞)	9	6	0.75	15001	4652975	4
ef14	(10, ∞)	11	7	1.00	37	1110	0
ef15	(10, ∞)	13	9	0.98	357	8801	0
ef21	(15, ∞)	5	11	0.89	71	1907	0
ef22	(15, ∞)	7	12	0.84	1783	74729	113
ef23	(15, ∞)	9	9	0.75	885	26960	391
ef24	(15, ∞)	11	-	-	-	-	-
ef25*	(15, ∞)	13	12	0.83	15002	70845	14
ef31*	(20, ∞)	5	18	0.83	15001	620127	0
ef32	(20, ∞)	7	15	0.85	1304	3095	0
ef33	(20, ∞)	9	-	-	-	-	-
ef34	(20, ∞)	11	-	-	-	-	-
ef35	(20, ∞)	13	-	-	-	-	-

Tabela 6.3: Resultados para as instâncias *ngcuts*, *cgcuts* e $ef's$ com restrições que apóiam a estabilidade.

Instância	$B=(L,C)$	N. de Itens	Altura da Faixa	Área (%)	Tempo (s)	N. de Nós Explorados	N. de Cortes
cgcut01	(15, ∞)	7	8	0.82	642	46557	0
cgcut02	(40, ∞)	10	–	–	–	–	–
cgcut03	(40, ∞)	20	–	–	–	–	–
ngcut1	(10, ∞)	5	11	0.86	2	112	0
ngcut2	(10, ∞)	7	11	0.61	65	6296	0
ngcut3	(10, ∞)	10	–	–	–	–	–
ngcut4	(15, ∞)	5	8	0.84	98	10363	3
ngcut5*	(15, ∞)	7	13	0.64	15012	525599	4
ngcut6	(15, ∞)	10	–	–	–	–	–
ngcut7	(20, ∞)	5	9	0.81	9	256	6
ngcut8*	(20, ∞)	7	19	0.71	15004	165847	47
ngcut09	(20, ∞)	10	–	–	–	–	–
ngcut10	(30, ∞)	5	–	–	–	–	–
ngcut11	(30, ∞)	7	–	–	–	–	–
ef11	(10, ∞)	5	8	0.95	4	51	0
ef12	(10, ∞)	7	7	0.93	39	3552	0
ef13	(10, ∞)	9	6	0.75	187	15169	14
ef14	(10, ∞)	11	7	1.00	86	4730	0
ef15	(10, ∞)	13	9	0.98	786	24131	0
ef21	(15, ∞)	5	11	0.89	50	2268	0
ef22	(15, ∞)	7	12	0.84	722	17033	3
ef23	(15, ∞)	9	9	0.75	4634	167034	6
ef24*	(15, ∞)	11	14	0.84	15072	273143	0
ef25*	(15, ∞)	13	12	0.83	15029	128429	26
ef31	(20, ∞)	5	18	0.83	1459	54436	0
ef32	(20, ∞)	7	15	0.85	1706	23434	2
ef33	(20, ∞)	9	–	–	–	–	–
ef34	(20, ∞)	11	–	–	–	–	–
ef35	(20, ∞)	13	–	–	–	–	–

Tabela 6.4: Resultados para as instâncias *ngcuts*, *cgcuts* e $ef's$ executando heurísticas.

Instância	N. de Itens	$B=(L,C)$	Área (%)	Altura HM	Altura HT	Altura HTM	Altura HPC	Altura HR	Altura HTR
cgcut1.txt	7	(15,8)	0.8167	8	8	17	9	17	9
cgcut2.txt	10	(40,84)	0.6512	92	114	143	84	143	84
cgcut3.txt	20	(40,527)	0.6421	551	551	527	527	527	527
ngcut1.txt	5	(10,11)	0.8636	24	11	24	24	24	24
ngcut2.txt	7	(10,16)	0.6080	19	16	28	18	28	18
ngcut3.txt	10	(10,14)	0.8095	19	14	32	22	32	22
ngcut4.txt	5	(15,8)	0.8438	11	11	8	8	8	8
ngcut5.txt	7	(15,15)	0.6351	15	15	22	22	22	22
ngcut6.txt	10	(15,13)	0.7949	24	16	34	13	34	13
ngcut7.txt	5	(20,9)	0.8148	9	12	18	9	18	9
ngcut8.txt	7	(20,22)	0.7091	29	22	29	29	29	29
ngcut9.txt	10	(20,42)	0.6619	63	42	58	58	58	58
ngcut10.txt	5	(30,32)	0.7188	55	32	55	55	55	55
ngcut11.txt	7	(30,33)	0.7566	61	33	61	61	61	61
ef11.txt	5	(10,8)	0.9500	15	8	13	13	13	13
ef12.txt	7	(10,7)	0.9286	14	7	11	11	11	11
ef13.txt	9	(10,5)	0.9000	8	7	6	6	5	5
ef14.txt	11	(10,7)	1.0000	15	7	12	12	12	12
ef15.txt	13	(10,9)	0.9778	19	9	14	14	14	14
ef21.txt	5	(15,11)	0.8909	15	11	17	17	17	17
ef22.txt	7	(15,12)	0.8389	12	14	15	15	15	15
ef23.txt	9	(15,9)	0.7481	11	9	10	10	10	10
ef24.txt	11	(15,12)	0.9778	22	22	14	14	12	12
ef25.txt	13	(15,12)	0.8333	16	12	18	18	18	18
ef31.txt	5	(20,18)	0.8333	18	18	18	18	18	18
ef32.txt	7	(20,15)	0.8467	18	15	38	18	38	18
ef33.txt	9	(20,24)	0.7000	24	24	24	24	24	24
ef34.txt	11	(20,24)	0.7354	24	24	33	24	33	24
ef35.txt	13	(20,24)	0.7458	24	24	34	24	34	24

Tabela 6.5: Resultados para as instâncias *ngcuts*, *cgcuts* e $ef's$ executando a heurística quase exata.

Instância	N. de Itens	$B=(L,C)$	Altura hqeSemApoio	Tempo (s) hqeSemApoio	Altura hqeComApoio	Tempo (s) hqeComApoio
cgcut1.txt	7	(15,8)	8	75	8	17
cgcut2.txt	10	(40,92)	–	–	–	–
cgcut3.txt	20	(40,527)	–	–	–	–
ngcut1.txt	5	(10,11)	7	5	7	7
ngcut2.txt	7	(10,16)	11	108	11	305
ngcut3.txt	10	(10,14)	12	15002	12	15001
ngcut4.txt	5	(15,8)	8	2	8	2
ngcut5.txt	7	(15,15)	13	333	13	172
ngcut6.txt	10	(15,16)	–	15003	13	15002
ngcut7.txt	5	(20,9)	9	3	9	2
ngcut8.txt	7	(20,22)	18	865	18	583
ngcut9.txt	10	(20,42)	–	–	–	–
ngcut10.txt	5	(30,32)	–	–	–	–
ngcut11.txt	7	(30,33)	–	–	–	–
ef11.txt	5	(10,8)	–	–	–	–
ef12.txt	7	(10,7)	7	14	7	40
ef13.txt	9	(10,6)	6	132	6	874
ef14.txt	11	(10,7)	7	131	7	431
ef15.txt	13	(10,9)	–	–	–	–
ef21.txt	5	(15,11)	9	21	9	88
ef22.txt	7	(15,12)	12	145	12	524
ef23.txt	9	(15,9)	7	3003	7	10532
ef24.txt	11	(15,12)	–	–	–	–
ef25.txt	13	(15,12)	–	–	–	–
ef31.txt	5	(20,18)	16	3	16	3
ef32.txt	7	(20,15)	14	2478	14	2704
ef33.txt	9	(20,24)	–	–	–	–
ef34.txt	11	(20,24)	–	–	–	–
ef35.txt	13	(20,24)	–	–	–	–

Tabela 6.6: Resultados para as instâncias da primeira categoria.

Instância	N. de Itens	$B=(L,C)$	Área (%)	Altura HM	Altura HT	Altura HTM	Altura HPC	Altura HR	Altura HTR
10/ins1.txt	10	(50,271)	0.6100	300	300	271	271	271	271
10/ins2.txt	10	(50,300)	0.5903	300	300	300	300	300	300
10/ins3.txt	10	(50,280)	0.5718	280	280	280	280	280	280
10/ins4.txt	10	(50,285)	0.5869	285	285	285	285	285	285
10/ins5.txt	10	(50,262)	0.5592	262	262	262	262	262	262
10/ins6.txt	10	(50,245)	0.6452	274	274	245	245	245	245
10/ins7.txt	10	(50,288)	0.5465	288	288	288	288	288	288
10/ins8.txt	10	(50,301)	0.5637	301	301	301	301	301	301
10/ins9.txt	10	(50,285)	0.5390	285	285	285	285	285	285
10/ins10.txt	10	(50,297)	0.5721	297	297	297	297	297	297
15/ins1.txt	15	(50,405)	0.5850	429	429	405	405	405	405
15/ins2.txt	15	(50,391)	0.6057	422	422	391	391	391	391
15/ins3.txt	15	(50,405)	0.5840	433	433	405	405	405	405
15/ins4.txt	15	(50,485)	0.6200	485	485	485	485	485	485
15/ins5.txt	15	(50,383)	0.5791	406	406	383	383	383	383
15/ins6.txt	15	(50,393)	0.6178	393	393	393	393	393	393
15/ins7.txt	15	(50,404)	0.6149	404	404	404	404	404	404
15/ins8.txt	15	(50,451)	0.6363	451	451	451	451	451	451
15/ins9.txt	15	(50,336)	0.6199	379	379	336	336	336	336
15/ins10.txt	15	(50,360)	0.6186	405	405	360	360	360	360
20/ins1.txt	20	(50,473)	0.6590	572	572	473	473	473	473
20/ins2.txt	20	(50,466)	0.6455	565	565	466	466	466	466
20/ins3.txt	20	(50,487)	0.6563	552	552	487	487	487	487
20/ins4.txt	20	(50,528)	0.6082	560	560	528	528	528	528
20/ins5.txt	20	(50,593)	0.6005	593	593	593	593	593	593
20/ins6.txt	20	(50,512)	0.5928	566	566	512	512	512	512
20/ins7.txt	20	(50,499)	0.6555	578	578	499	499	499	499
20/ins8.txt	20	(50,495)	0.5999	541	541	495	495	495	495
20/ins9.txt	20	(50,470)	0.6609	539	539	470	470	470	470
20/ins10.txt	20	(50,508)	0.6132	566	566	508	508	508	508

Tabela 6.7: Resultados para as instâncias da segunda categoria.

Instância	N. de Itens	$B=(L,C)$	Área (%)	Altura HM	Altura HT	Altura HTM	Altura HPC	Altura HR	Altura HTR
10/ins1.txt	10	(15,18)	0.7000	22	22	18	18	18	18
10/ins2.txt	10	(15,16)	0.7250	20	20	16	16	16	16
10/ins3.txt	10	(15,15)	0.7200	21	21	15	15	15	15
10/ins4.txt	10	(15,14)	0.7667	23	23	14	14	14	14
10/ins5.txt	10	(15,20)	0.7000	25	25	20	20	20	20
10/ins6.txt	10	(15,16)	0.7625	22	20	16	16	16	16
10/ins7.txt	10	(15,18)	0.6963	22	22	18	18	18	18
10/ins8.txt	10	(15,20)	0.7467	26	26	20	20	20	20
10/ins9.txt	10	(15,16)	0.7583	21	21	16	16	16	16
10/ins10.txt	10	(15,16)	0.7500	22	22	16	16	16	16
15/ins1.txt	15	(50,120)	0.7775	165	165	120	120	120	120
15/ins2.txt	15	(50,170)	0.6624	180	180	170	170	170	170
15/ins3.txt	15	(50,125)	0.7848	170	170	125	125	125	125
15/ins4.txt	15	(50,130)	0.7062	150	150	130	130	130	130
15/ins5.txt	15	(50,130)	0.6823	160	160	130	130	130	130
15/ins6.txt	15	(50,115)	0.7461	170	170	115	115	115	115
15/ins7.txt	15	(50,150)	0.6560	160	160	150	150	150	150
15/ins8.txt	15	(50,135)	0.7081	165	165	135	135	135	135
15/ins9.txt	15	(50,120)	0.8517	180	180	120	120	120	120
15/ins10.txt	15	(50,110)	0.8164	165	165	110	110	110	110
20/ins1.txt	20	(50,80)	0.7230	108	108	80	80	80	80
20/ins2.txt	20	(50,76)	0.7526	96	96	76	76	76	76
20/ins3.txt	20	(50,76)	0.7432	100	100	76	76	76	76
20/ins4.txt	20	(50,64)	0.7750	92	92	64	64	64	64
20/ins5.txt	20	(50,72)	0.8044	100	100	72	72	72	72
20/ins6.txt	20	(50,68)	0.7612	96	96	68	68	68	68
20/ins7.txt	20	(50,84)	0.7133	100	100	84	84	84	84
20/ins8.txt	20	(50,56)	0.8557	88	88	56	56	56	56
20/ins9.txt	20	(50,80)	0.7380	100	100	80	80	80	80
20/ins10.txt	20	(50,76)	0.7579	100	100	76	76	76	76

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho trata de uma variante do problema de Empacotamento em Faixa que considera uma ordem para os itens e requer que o empacotamento seja estável, segundo as condições de equilíbrio estático. O estudo da estabilidade foi desenvolvido usando conceitos e formulações presentes para a análise de vigas. Para o problema de Empacotamento em Faixa com Restrição de Ordem, apresentamos um modelo de programação linear inteira. Através de uma estratégia *branch-and-cut* resolvemos o modelo inteiro e adicionamos cortes relacionados à estabilidade do empacotamento.

A metodologia proposta para análise da estabilidade verifica a estabilidade do empacotamento analisando individualmente a estabilidade de cada item deste empacotamento. Através desta metodologia torna-se bem simples adicionar outras fontes de força/carga no empacotamento, como é o caso da forças externas que podem atuar nos itens. Como exemplo destas forças externas podemos citar a ação do vento, a inclinação do recipiente e o movimento do recipiente dentro de um caminhão.

Como a metodologia é independente do problema, é possível estudar a estabilidade dentro de qualquer problema que envolva corte e empacotamento, abrindo uma nova gama de problemas que podem ser investigados.

A estratégia *branch-and-cut* proposta é eficaz, porém a formulação linear inteira para o problema 2EEFO mostrou-se bastante lenta para instâncias de médio a grande porte.

As heurísticas tiveram bom desempenho e podem ser usadas para situações que envolvem um número maior de itens ou em situações que a precisão da altura do empacotamento não seja prioridade do problema.

Como trabalho futuro, pretendemos analisar empacotamentos que sofrem influência de forças externas. Estender a formulação proposta para lidar com a versão tridimensional do problema em questão, incluindo uma metodologia que lida com a estabilidade no espaço

tridimensional. Adicionar restrições que consideram o tamanho do braço do equipamento utilizado para retirar os itens do empacotamento. E, continuar estudando a discretização de pontos.

Referências Bibliográficas

[1] N. Bansal, X. Han, K. Iwama, M. Sviridenko, and G. Zhang. Harmonic algorithm for 3-dimensional strip packing problem. In *Proc. of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1197–1206, 2007.

[2] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.

[3] J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[4] J. E. Beasley. A population heuristic for constrained two-dimensional non-guillotine cutting. *European J. Operational Research*, 156:601–627, 2004.

[5] F. P. Beer and E. R. Johnston. *Resistência dos Materiais*. Makron Books, 1995.

[6] F. A. Campanari. *Teoria das estruturas*. Editora Guanabara Dois S.A., 1985.

[7] A. Caprara and M. Monaci. On the two-dimensional knapsack problem. *Operations Research Letters*, 32:5–14, 2004.

[8] J. N. Cernica. *Strength of Materials*. Holt, Rinehart and Winston, inc., 1966.

[9] C. H. Cheng, S. M. LEE, and Q. S. SHEN. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68–76, 1995.

[10] N. Christofides and E. Hadjiconstantinou. An exact algorithm for orthogonal 2-d cutting problems using guillotine cuts. *European J. Operational Research*, 83:21–38, 1995.

[11] N. Christofides and C. Whitlock. An algorithm for two dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[12] G. F. Cintra, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, 191:59–83, 2008.

[13] 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, editors,

Algorithms design for computer system design, pages 49–106. Springer-Verlag, New York, 1984.

[14] 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.

[15] A. L. Corcoran and R. L. Wainwright. A genetic algorithm for packing in three dimensions. *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pages 1021–1030, 1992.

[16] J. L. de Castro Silva, N. Y. Soma, and N. Maculan. A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions in Operational Research*, 10(2):141–153, 2003.

[17] K. A. Dowsland and W. B. Dowsland. Packing problems. *European J. Operational Research*, 56:2–14, 1992.

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

[19] H. P. Gavin. The three-moment equation for continuous beam. Technical Report, Department of Civil and Environmental Engineering, 2009.

[20] J. E. Gere and S. P. Timoshenko. Mecânica dos sólidos. *Livros Têcninos e Científicos Editora*, 1:256, 1983.

[21] J. M. Gere. *Mecânica dos Materiais*. 2003.

[22] P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

[23] E. Hadjiconstantinou and M. Iori. A hybrid genetic algorithm for the two-dimensional single large object placement problem. *European Journal of Operational Research*, 183:1150–1166, 2007.

[24] D. Halliday, R. Resnick, and J. Walker. *Fundamentos de Física: Mecânica*. LTC, 2006.

[25] R. C. Hibbeler. *Estática: Mecânica para Engenharia*. Prentice Hall Brasil, 2004.

[26] M. Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, 25:925–940, 1998.

[27] K. Jansen and R. Solis-Oba. An asymptotic approximation algorithm for 3D-strip packing. In *Proc. of the 17th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 143–152, 2006.

[28] K. Jansen and R. van Stee. On strip packing with rotations. In *Proc. of the 37th ACM Symposium on Theory of Computing*, 2005.

[29] L. Junqueira, R. Morabito, and D. S. Yamashita. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers and Operations Research*, In Press, Corrected Proof, 2010.

[30] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[31] C. Kenyon and E. Rémila. Approximate strip packing. In *37th Annual Symposium on Foundations of Computer Science*, pages 31–36, 1996.

[32] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European J. Operational Research*, 141(2):241–252, 2002.

[33] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8:363–379, 2004.

[34] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Oper. Res.*, 48(2):256–267, 2000.

[35] R. Morabito and M. Arenales. Abordagens para o problema do carregamento de contêineres. *Pesquisa Operacional*, 17(1):29–56, 1997.

[36] W. Nash. *Resistência dos Materiais*. McGraw Hill Brasil, 1990.

[37] D. Pisinger, O. Faroe, and M. Zachariasen. Guided local search for the three dimensional bin packing problem. *Techinical Report, Datalogisk institute*, 1999.

[38] W. F. Riley, L. D. Sturges, and D. H. Morris. *Mecânica dos Materiais*. Editora LTC, 2003.

[39] S. P. Timoshenko. *Resistência dos Materiais*. Livros Técnicos e Científicos, 1982.

[40] T. van Langendonck. *Vigas Simples Isostáticas*. Editora Científica, 1955.

[41] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.