Universidade Estadual de Campinas

Instituto de Matemática, Estatística e Computação Científica

Departamento de Matemática Aplicada

Modelo Não Linear para Minimizar o Número de Objetos Processados e o Setup num Problema de Corte Unidimensional

Luiz Leduíno de Salles Neto

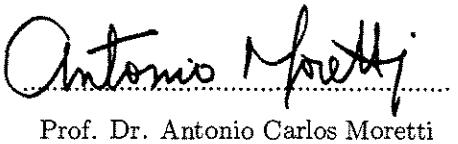
Orientador: Prof. Dr. Antonio Carlos Moretti

10 de junho de 2005

Modelo Não Linear para Minimizar o Número de Objetos Processados e o Setup num Problema de Corte Unidimensional

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por Luiz Leduíno de Salles Neto e aprovada pela comissão julgadora.

Campinas, 23 de junho de 2005



Prof. Dr. Antonio Carlos Moretti
Orientador

Banca Examinadora

Prof. Dr. Antonio Carlos Moretti (IMECC/UNICAMP)

Profa. Dra. Márcia Aparecida Gomes Ruggiero (IMECC/UNICAMP)

Prof. Dr. Aurélio Ribeiro Leite de Oliveira (IMECC/UNICAMP)

Prof. Dr. Marcos Nereu Arenales (ICMC/USP)

Prof. Dr. Reinaldo Morábito Neto (DEP/UFSCar)

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica, UNICAMP, como requisito parcial para obtenção do Título de DOUTOR em Matemática Aplicada.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Bibliotecário: Maria Júlia Milani Rodrigues – CRB8a / 2116

Salles Neto, Luiz Leduíno de

Sa34m Modelo não-linear para minimizar o número de objetos processados e o setup num problema de corte unidimensional / Luiz Leduíno de Salles Neto -- Campinas, [S.P. :s.n.], 2005.

Orientador : Antônio Carlos Moretti

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Problema de corte de estoque. 2. Heurística. 3. Programação não-linear. I. Moretti, Antônio Carlos. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Título em inglês: Nonlinear model to minimize both the number of processed objects and the number of setups in an cutting stock problem

Palavras-chave em inglês (Keywords): 1. Cutting stock problems. 2. Heuristic. 3. Nonlinear programming

Área de concentração: Pesquisa operacional

Titulação: Doutor em Matemática Aplicada

Banca examinadora: Prof. Dr. Antônio Carlos Moretti (UNICAMP)
Profa. Dra. Márcia Aparecida Gomes Ruggiero (UNICAMP)
Prof. Dr. Aurélio Ribeiro Leite de Oliveira (UNICAMP)
Prof. Dr. Marcos Nereu Arenales (USP)
Prof. Dr. Reinaldo Morábito Neto (UFSCar)

Data da defesa: 10/06/2005

Tese de Doutorado defendida em 10 de junho de 2005 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.

Prof. (a). Dr (a). ANTONIO CARLOS MORETTI

Prof. (a). Dr (a). MÁRCIA APARECIDA GOMES RUGGIERO

Prof. (a). Dr (a). AURÉLIO RIBEIRO LEITE DE OLIVEIRA

Prof. (a). Dr (a). MARCOS NEREU ARENALES

Prof. (a) Dr. (a) REINALDO MORÁBITO NETO

Aos meus pais, Carmen e Benedito.

Resumo

Neste trabalho apresentamos um novo método para minimizar o número de objetos processados e o número de padrões distintos (*setup*) num problema de corte unidimensional. Suavizamos a função objetivo, inteira e não linear, proposta por Haessler em 1975. Para gerar os padrões de corte utilizamos inicialmente uma heurística (SHP desenvolvida por Haessler), e posteriormente adaptamos o método de geração de colunas de Gilmore e Gomory para este modelo não-linear.

Palavras-Chaves: Problema de corte de estoque; Geração de colunas; Setup; Heurística; Programação Não-Linear.

Abstract

In this work we introduce a new method to minimize both the number of processed objects and the number of nonzeros cutting patterns (i.e., setup) in an one-dimensional cutting stock problem. To do so, we smooth the discontinuous nonlinear function used in Haessler(1975) to represent both objectives: the number of objects and setup number. To generate the cutting patterns we use the Gilmore&Gomory strategy with a starting basis given by the method SHP (Sequential Heuristic Procedure) developed by Haessler.

Keywords: Cutting stock problem; Column generation; Heuristic; Setup; Nonlinear programming.

Conteúdo

Prefácio

DA ETERNA PROCURA - Mário Quintana

Só o desejo inquieto, que não passa,
Faz o encanto da coisa desejada...
E terminamos desdenhando a caça
Pela doida aventura da caçada.

Apesar de não ser comum escrever um prefácio numa tese de doutorado, apenas uma seção de agradecimentos, optei em fazê-lo, pois achei interessante descrever, mesmo que brevemente, a importância da trajetória que culmina, por ora, neste trabalho.

Acredito ser importante registrar que, no início da graduação, desejava optar por matemática aplicada ao final do terceiro semestre do curso. Contudo, em virtude de uma bolsa de iniciação científica, oferecida pelo professor José Carlos de Souza Kill para estudar topologia, optei por matemática pura. Fiquei tão encantado com a matemática que acabei fazendo mestrado na área de Aplicações Harmônicas, também na matemática pura, com a orientação do Professor Caio Negreiros. Todavia, ao final do mestrado sentia que não era isso que eu buscava. Resolvi, então, entrar no doutorado em matemática aplicada, mas sem ainda saber o que fazer. Gentilmente o professor José Mário Martinez me ofereceu um projeto na área de programação não-linear, o qual aceitei e recebi apoio da FAPESP. Mas não era isso ainda o que me impulsionava

à pesquisa. Quando vi pela primeira vez Programação Linear tive certeza que era por ali que eu queria seguir. E por sorte, fiz esta disciplina com o professor Antonio Carlos Moretti, que além de profundo conhecimento passava um contagiante entusiasmo com a disciplina. Consultei os professores Martinez e Moretti sobre meu desejo de mudar de tema e os dois foram extremamente compreensíveis.

Como nem tudo são flores, pois se fossem, essas não seriam tão especiais, uma pedra apareceu no meio do caminho. Fiquei um ano afastado da universidade. Mesmo assim Moretti continuou disposto a me orientar. Aqui já é imprescindível registrar minha enorme gratidão por este gesto. Quando retornei ao doutorado, Moretti me apresentou uma idéia: resolver uma sequência de problemas suavizados para minimizar o desperdício e o *setup* num problema de corte unidimensional, utilizando para iniciar o trabalho a formulação proposta por Haessler [23] e o artigo do Martinez [39] sobre suavização de funções descontínuas. Depois de um ano, período em que cursei as disciplinas obrigatórias e fiz o primeiro exame de qualificação, iniciei de fato a pesquisa. Após estudar os artigos e fazer alguns simples testes computacionais, surge a primeira grande dificuldade: como adaptar a genial idéia de Gilmore e Gomory para um problema de programação não-linear? Até obter a proposta que apresentamos no capítulo 4 foram vários e vários meses de estudos e testes. Paralelamente a esta busca, tive que superar a falta de experiência com a implementação computacional, visto que minha formação em matemática pura restringiu meus conhecimentos em computação a duas disciplinas no primeiro ano de graduação. Novamente devo aqui agradecer ao Moretti pela paciência, pela sábia orientação e por ter acreditado que eu poderia superar essa e outras deficiências. E assim seguimos na "caçada", ora tentando uma linha, ora outra, ora concluindo que estava pronto, ora desanimando um pouco, mas sempre aprendendo algo.

Nestes dois anos de pesquisa foram importantes as apresentações dos resultados parciais na VII e VIII Oficina Nacional de Problemas de Corte e Empacotamento,

encontro anual do Projeto Temático da FAPESP "Teoria e Prática em Problemas de Corte e Empacotamento", e no XXVII Congresso Nacional de Matemática Aplicada. Nestes congressos houve várias sugestões úteis para a sequência do trabalho. Vale destacar, em particular, e agradecer as sugestões dos professores Marcos Nereu Arenales (ICMC/USP), Reinaldo Morábito (DEP/UFSCAR) e Horácio Hideki Yanasse (LAC/INPE).

Acredito que já é possível compreender o papel fundamental desta trajetória na minha convicta opção em seguir a vida acadêmica. Esta "eterna procura" é tão apaixonante que acaba superando, ao meu ver, um certo desalento quando vemos a injusta defasagem de recursos financeiros, notória quando comparamos a carreira docente com equivalentes na indústria ou no sistema bancário e financeiro.

Por fim, gostaria de agradecer a todos que contribuíram para esta conclusão. Não poderia começar sem citar meus pais, Benedito e Carmen, que dedicaram a vida para educar, em todos os sentidos, a mim e a minha irmã Camila, que também sempre esteve ao meu lado. Ofereço a eles este trabalho como uma singela forma de agradecer todo o amor que sempre me deram. Sou muito grato também à Kátia, pelo imenso amor, carinho, companheirismo e pelos momentos mágicos que, por consequência, tornaram mais agradáveis, criativos e produtivos estes anos de trabalho. Agradeço à Rosinha, Marília, Dantas, Gediel e Celsão pela imprescindível amizade. Sou grato aos colegas e amigos de trabalho na Universidade Estadual de Santa Cruz (UESC), especialmente ao Adson, Brígida, Gleidson, João Paulo e Luiz Antônio, que, pela amizade rápida e sinceramente construída, foram fundamentais para minha adaptação em Ilhéus e, assim, importantes para conclusão deste trabalho, desenvolvido durante um ano na "boa terra". Agradeço de forma geral a todos os demais familiares e amigos pelo constante apoio. Seria enfadonho, para quem lê, citar todos. Gostaria de agradecer também às professoras Marcia Aparecida Gomes Ruggiero e Valéria de Podestá Gomes, pelas valiosíssimas correções e sugestões dadas no segundo exame de qualificação. Pelos

mesmos motivos sou grato aos professores que formaram a banca examidora desta tese. E mais uma vez, agradeço ao Moretti, pelo que já disse acima, e pela amizade, dedicação, compreensão e incentivo.

É importante agradecer, ainda, às instituições que viabilizaram este trabalho e esta trajetória: à Unicamp, especialmente aos ótimos professores que tive, funcionários e colegas do IMECC; à CAPES, que me financiou através de sua bolsa por dois anos; à UESC, que me deu dignas condições de pesquisa e docência e uma licença remunerada de seis meses, após um ano de trabalho, para esta conclusão, entre outros benefícios.

Abreviações e Notação

$\lfloor x \rfloor$ - maior inteiro menor ou igual a x

$\lceil x \rceil$ - menor inteiro maior ou igual a x

$\mathbb{N}$ - conjunto dos números naturais: $\{0, 1, 2, ...\}$

$\mathbb{Z}$ - conjunto dos números inteiros

$\mathbb{R}$ - conjunto dos números reais

A^t - matriz transposta de A

A^{-1} - matriz inversa de A

$\inf X$ - ínfimo do conjunto $X \subset \mathbb{R}$

$\sup X$ - supremo do conjunto $X \subset \mathbb{R}$

MTB2 - Algoritmo de Martello e Toth para resolver o Problema da Mochila Limitada

PCaP - Problema do Carregamento de Paletes

PCaPD - Problema do Carregamento de Paletes do Distribuidor

PCaPP - Problema do Carregamento de Paletes do Produtor

PCC - Problema de Carregamento de Contêineres

PCD - Problema de Corte unidimensional para minimizar o Desperdício

PCE - Problemas de Corte e Empacotamento

PCOP - Problema de Corte unidimensional para minimizar o número de Objetos Processados

PCOPD - Problema de Corte unidimensional para minimizar o número de Objetos Processados no caso de Diversos tipos de bobinas em estoque

PCOPL - Problema de Corte unidimensional para minimizar o número de Objetos Processados no caso de Limitação de estoque

PCOPR - Problema de Corte unidimensional para minimizar o número de Objetos Processados Relaxado

PCOPS - Problema de Corte unidimensional para minimizar o número de Objetos Processados e o *Setup*

PCOPSR - Problema de Corte unidimensional para minimizar o número de Objetos Processados e o *Setup* Relaxado

PEB - Problema de Empacotamento de *Bin's*

PLA1 - Problema Linear Auxiliar 1

PLA2 - Problema Linear Auxiliar 2

PM0-1 - Problema da Mochila 0 ou 1

PM0-1R - Problema da Mochila 0 ou 1 Relaxado

PMI - Problema da Mochila Ilimitada

PML - Problema da Mochila Limitada

PMM0-1 - Problema de Múltiplas Mochilas 0-1

PPL - Problema de Programação Linear

PR - Problema Residual

Lista de Tabelas

Capítulo 1

Introdução

Nosso objetivo neste trabalho foi resolver o Problema de Programação Inteira Não Linear para minimizar o número de objetos processados e o *setup* num Problema de Corte Unidimensional. Tal formulação foi proposta por Haessler em 1975 [23], mas não foi abordada nem por ele nem por qualquer outro pesquisador até o momento. Para atacar tal problema optamos por estudar, inicialmente, os resultados apresentados por Martinez [39] sobre aplicações de técnicas de programação não linear em funções diferenciáveis que aproximam funções descontínuas. E como a geração de padrões de corte constitui uma etapa crucial num problema de corte unidimensional, buscamos uma heurística para gerar os padrões iniciais e depois uma adaptação do método de Gilmore e Gomory [21, 22] para nosso modelo não-linear.

Objetivamos neste texto definir os principais conceitos e apresentar os resultados necessários para compreensão do novo método que propomos, bem como, dar uma idéia geral da importância e do desenvolvimento da pesquisa sobre problemas de corte unidimensional, principalmente quando se objetiva minimizar o *setup*. Procuramos, sobretudo, apresentar nosso método de maneira clara, de forma que este possa ser reproduzido por quem lê o trabalho, e aferir a qualidade do mesmo de maneira justa.

Acreditamos que seja necessário apenas conhecimentos básicos sobre otimização para um razoável entendimento do texto.

Visando cumprir estes objetivos formatamos o trabalho como segue. No segundo capítulo apresentamos uma introdução aos Problemas de Corte e Empacotamento (PCE), dando ênfase ao Problema de Corte Unidimensional, suas diversas formulações e aplicações. No capítulo 3 detalhamos o método de geração de colunas, proposto inicialmente por Gilmore e Gomory [21, 22]. Para isso, apresentamos também o método simplex, algumas heurísticas para geração de soluções iniciais, os principais métodos de arredondamento e alguns métodos de resolução do problema da mochila. No capítulo 4 descrevemos quatro métodos para minimizar o número de objetos processados e o *setup* num problema de corte unidimensional: SHP, KOMBI, *branch-cut-price*, ILS-APG. Damos especial atenção aos métodos SHP e KOMBI, pois os utilizamos para aferir a qualidade do novo método que propomos neste trabalho. A descrição deste método é feita no capítulo 5, após apresentarmos os resultados de Martinez [39] sobre suavização de funções custo descontínuas. No capítulo 6 detalhamos a implementação computacional, descrevemos os testes computacionais e os resultados obtidos. No capítulo 7 apresentamos nossas conclusões e perspectivas de trabalhos futuros. Escrevemos ainda um apêndice sobre as Classes $\mathcal{P}$ e $\mathcal{NP}$, que traz uma análise da complexidade computacional de alguns Problemas de Corte e Empacotamento. Ao final, nos apêndice B e C, transcrevemos, respectivamente, os pseudo-códigos do método *branch-and-bound* e de um gerador de soluções iniciais que utilizamos no novo método que propomos.

Vale registrar que utilizamos apenas *softwares* livres ou gratuitos para confecção deste trabalho: editor LaTeX2e; linguagem *Fortran*, com compilador g77 para os códigos; Mupad Light 2.5.3 para os gráficos; e sistema operacional Linux. Também gostaríamos de destacar que evitamos, na medida do possível, usar palavras de origem anglo-saxônica. Além de *softwares* citada acima, utilizamos as palavras *setup*, *bin*, *simulated annealing*, *branch-and-price*, *branch-cut-price*, *branch-and-bound*, *forward move*

e *bactracking move* no contexto do algoritmo *branch-and-bound*, pois são largamente utilizadas em todo o mundo, e não encontramos tradução destas em nenhum artigo ou livro. Também procuramos não usar anglicismos.

Capítulo 2

Problemas de Corte e Empacotamento

2.1 Introdução

O Problema de Corte consiste, basicamente, em encontrar a melhor maneira de obter peças de diferentes tamanhos a partir do corte de peças maiores. Mais especificamente, suponhamos que deseja-se obter m tipos de peças diferentes, cada uma com certa especificidade (comprimento, área, volume ou massa) w_i, em uma quantidade d_i, a partir de peças maiores de capacidade $W > w_i$, para $i = 1, ..., m$. O Problema de Corte objetiva encontrar a melhor forma de se cortar as peças maiores, minimizando algum tipo de custo ou maximizando o lucro, para se obter os itens desejados, nas quantidades demandadas.

Já o reverso desse problema, encontrar a melhor forma de armazenar m diferentes tipos de peças de especificidade w_i, numa quantidade d_i, num recipiente de capacidade W, ou seja, o problema de encontrar a melhor forma de armazenar peças menores num

recipiente maior, é chamado de Problema de Empacotamento. Devido a dualidade entre corte e espaço, o Problema de Corte poder ser visto como Problema de Empacotamento e vice-versa. Constituiu-se assim, um mesmo ramo da Pesquisa Operacional, chamado de Problemas de Corte e Empacotamento (PCE).

Kantorovich produziu em 1939 o primeiro trabalho, publicado em 1960 [30], com uma formulação para minimizar as perdas (*trim loss*) num problema de corte unidimensional. Problemas similares foram tratados por Paull e Walter em 1954 [44], Metzger em 1958 [41] e Eilon em 1960 [16]. Contudo, em todos eles foram apresentados métodos adequados apenas para problemas pequenos (Dowsland e Dowsland [13]). Com os trabalhos de Gilmore e Gomory em 1961 e 1963, [21] e [22] respectivamente, houve um significativo avanço no estudo de problemas de corte através do processo de geração de colunas, que estudaremos no terceiro capítulo.

Desde então o estudo dos problemas de corte ganhou grande importância para o planejamento da produção em algumas indústrias, como de papel, vidro, química, têxtil e metalúrgica. O objetivo geralmente é de minimizar as perdas, mas pode ser também maximizar o lucro, minimizar o número de objetos (matéria prima) usados, o tempo de produção ou uma combinação desses. Já o estudo dos problemas de empacotamento têm aplicações principalmente nas áreas de logística de diversas indústrias, no setor atacadista e nas empresas de transporte rodoviário, ferroviário, aéreo e marítimo. O objetivo geralmente é de maximizar o uso do espaço, ou seja, empacotar o maior número de peças no menor número de paletes, caixas, vagões, caminhões, contêineres, compartimentos de aviões...

Devido a este grande número de aplicações e a dificuldade de resolução, visto que a maioria dos PCE pertencem à classe $\mathcal{NP}$-completo (ver Apêndice A), vários pesquisadores em todo o mundo têm concentrado esforços no desenvolvimento de novos e eficientes métodos de resolução, em sua grande maioria heurísticos.

Com base nos diversos trabalhos publicados na literatura - mais de 700 até 1990, data da pesquisa - Dyckhoff [14] propôs uma classificação dos PCE através de quatro características principais, baseada na lógica do problema: dimensão, forma de alocação das unidades, sortimento de unidades grandes e sortimento de unidades pequenas. Essa classificação foi importante, pois permitiu concentrar as pesquisas sobre tipos de problemas melhor definidos. Cada uma destas características foram subdivididas nos seguintes tipos (representados pelos símbolos entre parênteses):

1. Dimensão:

   (1)Unidimensional;

   (2)Bidimensional;

   (3)Tridimensional;

   (N) N-dimensional, com $N > 3$.

2. Forma de Alocação das Unidades:

   (V) seleção de unidades grandes;

   (B) seleção de unidades pequenas.

3. Sortimento de Unidades Grandes:

   (O) uma unidade;

   (I) unidades de tamanhos iguais;

   (D) unidades de tamanhos diferentes.

4. Sortimento de Unidades Pequenas:

   (F) poucas unidades de tamanhos diferentes;

   (M) muitas unidades de muitos tamanhos diferentes;

   (R) muitas unidades de poucos tamanhos diferentes;

   (C) unidades de tamanhos iguais.

Vale notar que a combinação destas características resulta em 96 tipos diferentes de problemas. E cada um destes tipos pode apresentar restrições diversas.

A tipologia é denotada pela quadrúpla $\alpha/\beta/\gamma/\delta$, onde α respresenta a dimensão do problema, β a forma de alocação, γ o sortimento de unidades e δ o sortimento de unidades pequenas. Por exemplo, a tipologia $2/V/I/M$ representa um problema bidimensional, com seleção de unidades iguais, sortimento de unidades grandes iguais e muitas unidades de tamanhos diferentes. Na tabela 2.1 apresentamos alguns problemas clássicos e sua tipologia.

Problema	Tipologia
Problema de Corte de Estoque (clássico)	$1/V/I/R$
Problema da Mochila	$1/B/O/$
Problema de Empacotamento de *Bin's*	$1/V/I/M$
Problema de Carregamento de Veículos (clássico)	$1/V/I/F$
Problema do Carregamento de Paletes do Produtor	$2/B/O/C$
Problema do Carregamento de Paletes do Distribuidor	$2/B/O/$
	$2/V/I/$

Tabela 2.1: Exemplos da Tipologia de Dyckhoff

Quando alguma especificação na tipologia de um problema não é fixada, é porque esta não caracteriza, ou não altera a lógica do problema, ou mesmo porque existem várias formulações e restrições diferentes referentes a especificação em branco para o mesmo problema. Vale citar, para exemplificar, a tipologia $1/B/O/$ do Problema da Mochila que não especifica o tipo de sortimento de unidades pequenas. De fato, se houver poucas unidades de tamanhos diferentes, muitas unidades de muitos tamanhos diferentes ou muitas unidades de poucos tamanhos diferentes, não há alteração na formulação do problema. Na seção 2.3 abordaremos algumas formulações do Problema da Mochila que evidenciarão que estes três tipos de sortimentos de unidades pequenas são irrelevantes para a lógica do problema. Vale observar, por fim, que o Problema da

Mochila se torna trivial, e assim desinteressante, no caso de termos apenas unidades pequenas de tamanhos iguais.

Nas próximas seções abordaremos brevemente cada um dos problemas listados na tabela 2.1.

2.2 O Problema de Corte de Estoque Unidimensional

O Problema de Corte de Estoque Unidimensional $(1/V/I/R)$ é caracterizado, como o próprio nome sugere, pelo corte em apenas um sentido, ou seja, uma bobina mestre passa numa máquina que faz cortes em apenas uma dimensão. Mais especificamente, tem-se um número m de diferentes itens, cada um de largura w_i, que devem ser produzidos para atender cada uma das demandas d_i, a partir de uma bobina mestre de largura $W > w_i$ para todo i, através de cortes ao longo de seu comprimento.

Chamamos cada disposição dos itens na bobina mestre de "padrão de corte".

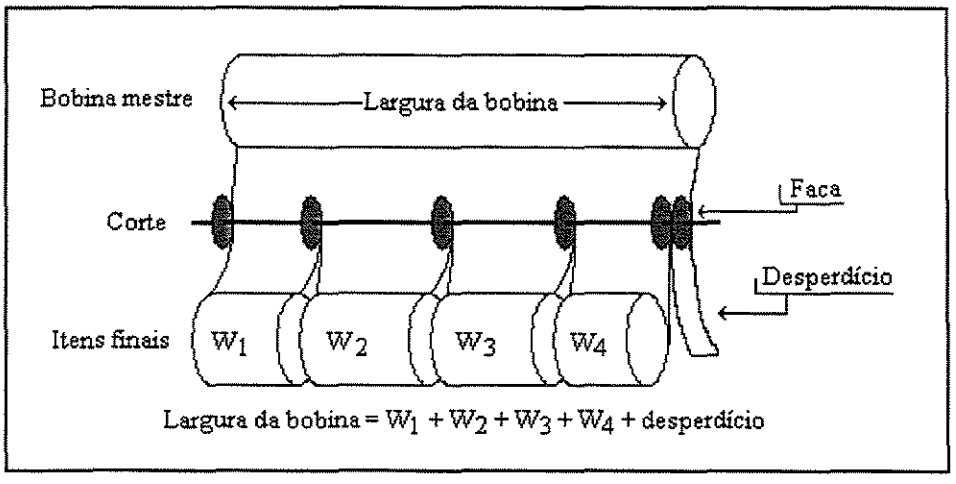


Figura 2.1: Processo de Corte Unidimensional

Vejamos um exemplo para clarear as idéias.

Suponhamos que uma indústria possua um número ilimitado de bobinas de 2 metros de largura no estoque e três tipos de itens, em três diferentes larguras, para serem produzidos a partir destas bobinas mestre: 30 centímetros(cm), 40 centímetros, 50 centímetros. Dentre todas as possibilidades de disposição dos itens em cada bobina (padrões de corte), listamos quatro:

1) 6 itens de 30 cm;

2) 5 itens de 30 cm e 1 item de 50 cm;

3) 2 itens de 40 cm e 2 itens de 50 cm;

4) 3 itens de 50 cm e 1 item de 40 cm.

Em cada padrão de corte calculamos o quanto é desperdiçado, isto é, a "sobra" (ver figura 2.2):

Perda no Padrão 1= 200 - 6x30= 20 cm;

Perda no Padrão 2= 200 - (5x30+50)=0;

Perda no Padrão 3= 200 - (2x40+2x50)=20 cm;

Perda no Padrão 4= 200 - (3x50+40)=10 cm.

O objetivo na indústria é óbvio: atender a demanda com menor custo possível. É razoável assumir como um custo relevante a quantidade de material desperdiçado, visto que podem haver, e provavelmente haverá, padrões de corte com "sobras".

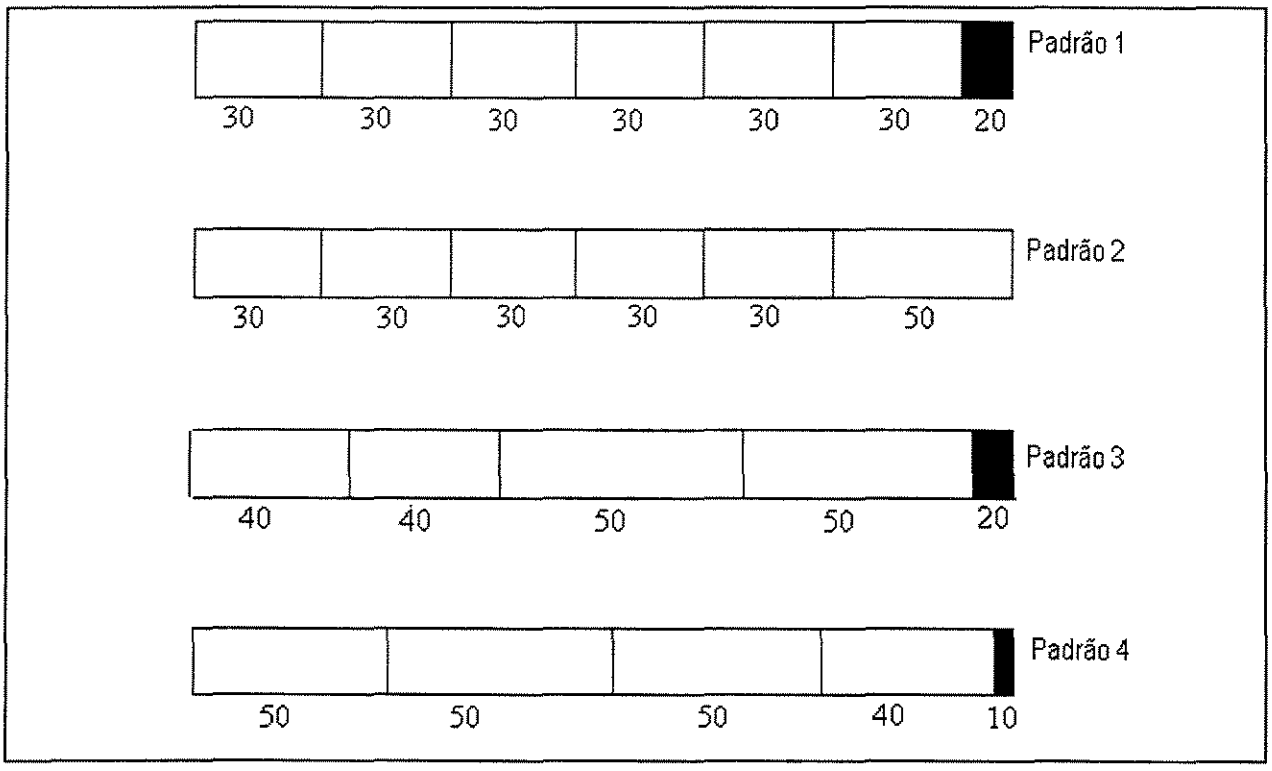


Figura 2.2: Padrões de Corte (desperdício em preto)

Voltando ao nosso exemplo, suponhamos que seja requisitado 100 unidades de 30 centímetros, 80 de 40 centímetros, 120 de 50 centímetros e os padrões de corte disponíveis são os quatro listados acima. Se definirmos x_i como o número de bobinas cortadas com o padrão i, temos o seguinte Problema de Programação Inteira:

$$
\begin{aligned}
\text{Minimizar} \quad & (20x_1 + x_2 + 20x_3 + 10x_4) \\
\text{sujeito a:} \quad & 6x_1 + 5x_2 = 100 \\
& 2x_2 + x_4 = 80 \\
& x_2 + 2x_3 + 3x_4 = 120 \\
& x \in \mathbb{N}
\end{aligned}
$$

De forma geral temos o seguinte problema de programação inteira que chamaremos de Problema de Corte Unidimensional para minimizar o Desperdício (PCD):

$$\begin{aligned} \text{Minimizar} \quad & c_1 \sum_{j=1}^{n} T_j x_j \\ \text{sujeito a:} \quad & \textstyle\sum_{j=1}^{n} a_{ij} x_j = d_i \quad i = 1, ..., m \\ & x_j \in \mathbb{N} \qquad\qquad j = 1, ..., n. \end{aligned}$$

onde:

n - número de possíveis padrões de corte;

c_1 - custo por metro desperdiçado;

x_i - número de bobinas processadas com o padrão de corte i;

$T_j = W - \sum_{i=1}^{m} a_{ij} w_i$ - perda em metros em cada padrão j;

a_{ij} - número de itens do tipo i no padrão j;

d_i - número de itens do tipo i demandados.

No exemplo acima algumas perguntas são pertinentes: por que escolhemos aqueles padrões de corte? E se escolhêssemos outros? Mais geralmente, é possível trabalhar com todos os padrões de corte viáveis? Do ponto de vista computacional não. Com efeito, o número de padrões de corte viáveis é geralmente muito grande, o que tornaria qualquer método de resolução muito lento. Desta forma, a escolha dos padrões a serem avaliados constitui uma etapa crucial na resolução de um Problema de Corte. Como

já observamos, no capítulo 3 daremos especial atenção a um processo de geração de padrões de corte, chamado de geração de colunas.

Podemos considerar como objetivo minimizar o número de bobinas utilizadas ao invés de minimizar o desperdício. Chamaremos o problema de corte com este objetivo de Problema de Corte Unidimensional para Minimizar o número de Objetos Processados(PCOP):

$$\begin{aligned} \text{Minimizar} \quad & c\sum_{j=1}^{n} x_j \\ \text{sujeito a:} \quad & \sum_{j=1}^{n} a_{ij}x_j = d_i, \quad i = 1, ..., m. \\ & x_j \in \mathbb{N}, \qquad\qquad j = 1, ..., n. \end{aligned}$$

onde c é o custo de cada bobina.

Porém, se exigimos que a demanda deva ser atendida exatamente, ou seja, o excesso de produção será considerado desperdício, é fácil se convencer, e mais rigorosamente demonstrar, que minimizar o número de bobinas utilizadas é equivalente a minimizar o desperdício.

Proposição 2.1. *Um vetor x é solução ótima do PCD se e somente se é também solução ótima do PCOP.*

Demonstração:

No PCD temos que minimizar a função $f(x) = c_1 \sum_{j=1}^{n} T_j x_j$.

Temos $T_j = W - \sum_{i=1}^{m} a_{ij} w_i$. Sem perda de generalidade podemos assumir que $c_1 = 1$.

Assim $f(x) = \sum_{j=1}^{n} (W - \sum_{i=1}^{m} a_{ij} w_i) x_j \Rightarrow f(x) = \sum_{j=1}^{n} (W x_j - \sum_{i=1}^{m} a_{ij} w_i x_j)$.

Donde $f(x) = W \sum_{j=1}^{n} x_j - \sum_{i=1}^{m} w_i \sum_{j=1}^{n} a_{ij} x_j$.

Como $\sum_{j=1}^{n} a_{ij} x_j = d_i$ temos $f(x) = W \sum_{j=1}^{n} x_j - \sum_{i=1}^{m} w_i d_i$.

Logo, x minimiza $\sum_{j=1}^{n} T_j x_j$ se e somente se minimiza $\sum_{j=1}^{n} x_j$, como queríamos demonstrar.

Podemos trabalhar ainda com outras restrições e objetivos, como veremos nas próximas subseções.

2.2.1 Diversos tipos de bobinas em estoque em quantidade ilimitada

Este caso, onde temos vários tipos de bobinas no estoque em quantidade ilimitada, ocorre em indústrias que produzem as bobinas em máquinas diferentes ou as adquirem no mercado em diferentes tamanhos, como as barras de aço na construção civil.

Assumindo que temos NB diferentes bobinas em estoque, cada uma de largura W_k e custo c_k, para cada $k = 1, ..., NB$, temos o seguinte Problema de Corte para Minimizar o número de Objetos Processados no caso de Diversos tipos de Bobinas em estoque (PCOPD):

$$\text{Minimizar} \quad \sum_{k=1}^{NB} c_k \sum_{j=1}^{n} x_{kj}$$

$$\text{sujeito a:} \quad \sum_{k=1}^{NB} \sum_{j=1}^{n} a_{ij} x_{kj} = d_i, \quad i = 1, ..., m.$$

$$x_{kj} \in \mathbb{N}, \qquad k = 1, ..., NB, \quad j = 1, ..., n.$$

onde:

x_{kj}= número de vezes que a bobina do tipo k é processada com o padrão j.

2.2.2 Diversos tipos de bobinas em estoque em quantidade limitada

Este caso ocorre em indústrias com limitada capacidade de produção das bobinas mestres, ou naquelas onde as bobinas são compradas com certa antecedência, devido à demora ou incerteza no prazo de entrega, como as bobinas de aço na indústria metalúrgica.

Seja b_k a disponibilidade em estoque da bobina do tipo k, $k = 1, ..., NB$. Temos o seguinte Problema de Corte para minimizar o número de Objetos Processados no caso de Limitação de estoque (PCOPL):

$$\text{Minimizar} \quad \sum_{k=1}^{NB} \sum_{j=1}^{n} x_{kj}$$

$$\text{sujeito a:} \quad \sum_{k=1}^{NB} \sum_{j=1}^{n} a_{ij} x_{kj} = d_i \qquad i = 1, ..., m.$$

$$\sum_{j=1}^{n} x_{kj} \leq b_k \qquad k = 1, ..., NB.$$

$$x_{kj} \in \mathbb{N}, \qquad k = 1, ..., NB, \;\; j = 1, ..., n.$$

2.2.3 Tolerância em relação à demanda

Existem ainda outros modelos que podem ser úteis e necessários no planejamento da produção em determinadas indústrias. Pode-se estabelecer, por exemplo, uma tolerância em torno da demanda solicitada, ou seja, a produção de cada item i pode ficar entre um valor mínimo d_i e um valor máximo D_i, dependendo exatamente da demanda, tempo de produção e do custo de estoque.

Correia e outros tratam em [10] dessa tolerância na indústria papeleira da Europa. Existe um Padrão de Tolerância Europeu (tabela 2.2) que deve ser respeitado quando há excesso de produção ou mesmo sub-produção. Mais especificamente, o cliente é obrigado a aceitar uma variação na quantidade produzida quando esta satisfaz a tolerância prevista na tabela. Quando o excesso de produção está acima dessa tolerância o Departamento de Vendas tenta negociar a aceitação desta quantidade extra com o cliente.

Quantidade Demandada (em toneladas)	Tolerância
Acima de 100	Prévio acordo
50-100	+/- 4%
20-50	+/- 6%
10-20	+/- 8%
5-10	+/- 10%
3-5	+/- 15%
Menos de 3	+/- 20%

Tabela 2.2: Padrão de Tolerância Europeu na Indústria de Papel

2.2.4 O *setup* num Problema de Corte

Voltando nossa análise para a função objetivo, vale observar que, em alguns casos, minimizar o número de objetos processados e/ou o deperdício pode não ser suficiente. De fato, quando, por exemplo, há uma demanda grande para ser atendida em pouco tempo, ganha substancial importância o custo de cada troca de padrão de corte, uma vez que estas trocas demandam ajuste de máquinas (facas para os cortes) e, por conseguinte, tempo (*setup*) e trabalho. Acrescentando este custo ao PCOP, obtemos o modelo para o Problema de Corte Unidimensional para Minimizar o número de Objetos Processados e o *Setup* (PCOPS):

$$\text{Minimizar} \quad c_1 \sum_{j=1}^{n} x_i + c_2 \sum_{j=1}^{n} \delta(x_j)$$

$$\text{sujeito a:} \quad \sum_{j=1}^{n} a_{ij} x_j = d_i, \qquad i = 1, ..., m.$$

$$x_j \in \mathbb{N}, \qquad j = 1, ..., n.$$

onde:

c_2 - custo de cada troca do padrão de corte;

$$\delta(x_j) = \begin{cases} 1 & \text{se } x_i > 0, \\ 0 & \text{se } x_i = 0. \end{cases}$$

Um outro exemplo simples da aplicabilidade deste tipo de modelo está presente nas indústrias gráficas, conforme Teghem e outros abordam em [48]: considere o problema de imprimir capas de livros num custo mínimo, e que em cada chapa matriz é possível colocar 4 capas, isto é, cada padrão comporta 4 capas. Existe um custo fixo para produzir cada chapa matriz e um custo relativamente pequeno para produzir cada folha a partir desta chapa matriz. Ou seja, o número de padrões usados é o que eleva realmente o custo de impressão das capas.

Problemas de otimização combinatória envolvendo custos de *setup* (custos fixos) são notoriamente difíceis de se resolver. No capítulo 4 faremos uma breve revisão da literatura sobre o PCOPS. No capítulo 5 apresentaremos um novo método de resolução baseado num modelo não linear que aproxima o PCOPS. No capítulo 6 detalharemos a implementação computacional do nosso método e os testes computacionais para aferir a qualidade do mesmo.

2.3 O Problema da Mochila

O Problema da Mochila ($1/B/O/$) apareceu na literatura em 1957 no artigo *"Discrete variable extremum problems"*, de George Dantizg e recebeu este nome devido à seguinte motivação :

Uma pessoa pretende viajar com uma Mochila de Capacidade W, em quilogramas

por exemplo, e precisa escolher quais utensílios levar, como calça, camisa, toalha, sapato, livro, alicate, botas ... Atribuindo a cada utensílio de peso w_i uma utilidade u_i, o que levar de forma a maximizar a utilidade total?

A modelagem deste Problema é simples: associamos a cada objeto uma variável x_i que assume 1 se o mesmo é colocado na mochila e 0 caso contrário. Obtemos a seguinte formulação para o Problema da Mochila 0-1 (PM0-1):

$$\text{Maximizar} \quad \sum_{i=1}^{n} u_i x_i$$

$$\text{sujeito a:} \quad \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\}, \qquad i = 1, ..., n.$$

onde n é o número de objetos.

Este modelo é apropriado também para a seguinte situação: suponha que você quer investir W reais e precisa escolher entre n possíveis investimentos, e cada um desses investimentos gera um lucro de u_i reais para um investimento de w_i reais, $i = 1, ..., n$. Não é permitido investir na aplicação i quantia diferente de w_i reais para cada $i = 1, ...n$, ou seja, ou se investe w_i reais na aplicação i ou não se investe na aplicação i.

Na seção 3.5 falaremos sobre alguns métodos de resolução do PM0-1, mas acreditamos ser instrutivo citarmos já uma heurística de construção gulosa que obtém, de forma muito simples e rápida, um solução para o PM0-1 de razoável qualidade em muitos casos.

Chamamos uma estratégia de gulosa quando esta leva em conta sobretudo o valor,

a grandeza, a utilidade daquilo que se está analisando. Não por acaso as estratégias gulosas também são chamadas de míopes, visto que "enxergam" apenas os grandes valores. Para clarear as idéias, vale citar uma simples heurística gulosa para o Problema da Mochila 0-1:

Heurística Gulosa para o PM0-1

- Seja W a capacidade da mochila, m o número de itens, cada um com uma utilidade u_i e peso w_i, tal que $u_1/w_1 > u_2/w_2 > ... > u_m/w_m$;
- Para i=1,...,m faça
  - Se $W > w_i$ coloque o tem i na mochila;
  - Faça $W = W - w_i$

Esta heurística é chamada gulosa pois coloca na mochila preferencialmente os itens com maior densidade (*utilidade/peso*). Para mais informações sobre as heurísticas gulosas para o PM0-1 veja a referência [38].

O PM0-1, contudo, não atende a todos os "problemas da mochila" que encontramos no dia a dia. De fato, em muitas situações é preciso definir a quantidade de objetos que serão colocados na mochila ao invés de simplesmente definir se o mesmo é simplesmente colocado ou não, ou a quantia que investiremos em cada tipo de aplicação. No exemplo da pessoa que arruma a mochila para viagem supracitado, pode ser mais útil e necessário definir quantas camisas e calças levar do que se leva uma unidade ou não. Para modelar este problema definimos x_i como sendo a quantidade de objetos do tipo i a serem levados na mochila. Temos a seguinte formulação desse que chamaremos de Problema da Mochila Ilimitada (PMI):

$$\text{Maximizar} \quad \sum_{i=1}^{n} u_i x_i$$

$$\text{sujeito a:} \quad \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \mathbb{N}, \qquad i = 1, ..., n.$$

Vale observar que o PMI é uma boa formulação para o problema de cortar itens menores de apenas uma bobina maior.

Outra restrição que pode aparecer em problemas reais é a limitação do número de objetos que podem ser colocados na Mochila. Voltando novamente ao nosso exemplo, para clarear as idéias, suponhamos que tenha-se à disposição 7 camisas, 5 calças e 2 livros para serem levados para viagem. Neste caso seria preciso alterar o modelo do problema, acrescentando estes limitantes às restrições. Obtemos assim o Problema da Mochila Limitada (PML):

$$\text{Maximizar} \quad \sum_{i=1}^{n} u_i x_i$$

$$\text{sujeito a:} \quad \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \leq b_i, \qquad i = 1, ..., n$$

$$x_i \in \mathbb{N} \qquad i = 1, ..., n.$$

onde b_i é o número máximo de objetos do tipo i que podem ser postos na mochila.

Vale observar que o PM0-1 pode ser transformado num PML fazendo $b_i = 1$ para

todo $i = 1, ...n$. Da mesma forma o PMI pode ser formulado como um PML fazendo $b_i = \lfloor W/w_i \rfloor$.

Uma importante generalização do Problema da Mochila 0-1 (PM-01) é o Problema de Múltiplas Mochilas 0-1 (PMM0-1), apropriado para o caso de haver m mochilas, cada uma de capacidade c_i. Temos o seguinte modelo do PMM0-1:

$$\text{Maximizar} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij}$$

$$\text{sujeito a:} \quad \sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i, \quad i = 1, ..., m$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad j = 1, ..., n$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, ..., m, \quad j = 1, ..., n.$$

onde $x_{ij} = 1$ se o item j é colocado na mochila i.

Se $m = 1$ o PMM0-1 se reduz, obviamente, ao PM0-1.

Os Problemas da Mochila tem várias aplicações em diversas áreas da Pesquisa Operacional. Por exemplo, em investimentos de capital, orçamento, carregamento de veículos e como um sub-problema em Problemas de Corte como veremos no capítulo 3. Em contrapartida são problemas de difícil resolução, visto serem problemas *$\mathcal{NP}$-difícil* (ver apêndice A).

2.4 Problema de Empacotamento de *Bin's*

Suponhamos agora a seguinte situação: uma fábrica produz n produtos diferentes, cada um de tamanho w_i, e possui um único tipo de recipiente, com capacidade W, em quilogramas por exemplo, para despachá-los. Deseja-se distribuir os produtos no menor número possível de recipientes (*bin's*). Podemos formular este problema, chamado de Empacotamento de *Bin's* (PEB), da seguinte maneira:

$$\text{Minimizar} \quad \sum_{i=1}^{n} y_i$$

$$\text{sujeito a:} \quad \sum_{j=1}^{m} w_j x_{ij} \leq W y_i, \quad i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1 = 1, ..., m$$

$$y_i \in \{0, 1\} \qquad i = 1, ..., n$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, ..., n, \; j = 1, ..., m.$$

onde $y_i = 1$ se o *bin* i é usado e 0 caso contrário; e $x_{ij} = 1$ se o item j é associado ao *bin* i e 0 caso contrário.

Esta formulação é ideal, também, no caso de se necessitar produzir um item de cada tipo de peça a partir de um único tipo de bobina mestre. Vale observar ainda que podemos "olhar" o PEB como um problema de várias mochilas (bins) 0 ou 1. Como vimos, o PEB tem a seguinte tipologia $1/V/I/M$. Assim, o que diferencia o PEB do Problema de Corte de Estoque Clássico é o sortimento de unidades pequenas: no PEB temos muitas unidades de muitos tamanhos diferentes enquanto no Problema de Corte

de Estoque temos muitas unidades de poucos tamanhos diferentes.

Vale destacar que o PEB é *$\mathcal{NP}$-difícil no sentido forte* (ver apêndice A).

2.5 Problema de Carregamento de Veículos (Clássico)

O Problema de Carregamento de Veículos tem a seguinte tipologia $(1/V/I/F)$ como vimos na tabela 2.1. Nota-se que a única diferença desse problema para o PEB se dá em relação ao sortimento de unidades pequenas, visto que neste caso temos poucas unidades de tamanho pequeno. Para clarear as idéias, suponhamos a seguinte situação comum em diversas empresas: deseja-se transportar n tipos de cargas, cada uma de peso w_i, no menor número possível de veículos de capacidade W. Eilon e Christofildes [17] propuseram a seguinte formulação para este problema:

$$\text{Minimizar} \quad \sum_{j=1}^{m} p_j \sum_{i=1}^{n} x_{ij}$$

$$\text{sujeito a:} \quad \sum_{i=1}^{n} w_i x_{ij}, \leq L \quad j = 1, ..., m$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, ..., n, \quad j = 1, ..., m$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, ..., n, \quad j = 1, ..., m.$$

onde:

$x_{ij} = 1$ se o item de carga i for alocado no veículo j e 0 caso contrário;

p_j é a penalidade associada à alocação de itens no veículo j.

Para atingir o objetivo de minimizar o número de carros utilizados, a penalidade p_j aumenta com j de forma a termos $p_{j+1} >> p_j > 0$. Com isso, é sempre melhor carregar veículos já ocupados com alguns itens antes de carregar um novo veículo ainda vazio.

2.6 Problema do Carregamento de Paletes

O Problema do Carregamento de Paletes (PCaP - $2/B//$) consiste, basicamente, em encontrar a melhor forma de empacotar caixas retangulares num palete (plataforma de madeira, metal, fibra ou outro material, disposta horizontalmente, na qual a carga pode ser empilhada e estabilizada). Geralmente se objetiva maximizar o número de caixas no palete.

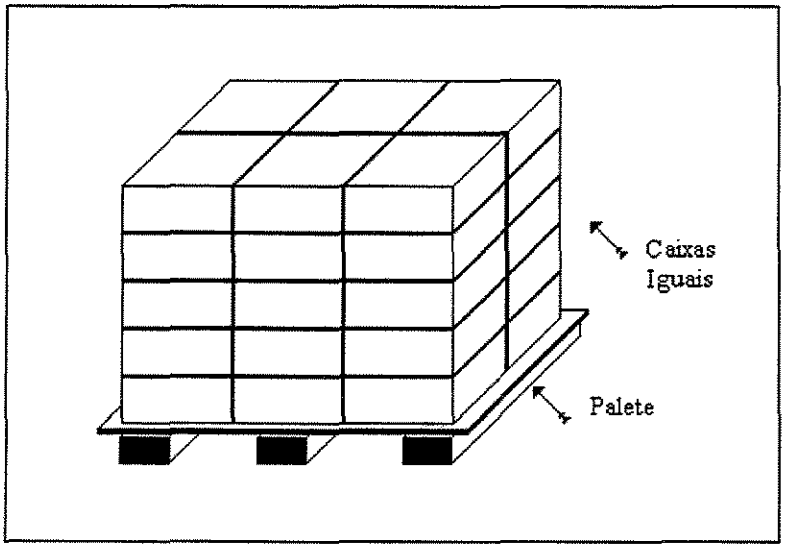


Figura 2.3: Palete do Produtor

Hodgson [27] dividiu o PCaP em dois casos (ver figuras 2.3 e 2.4):

1. PCaP do Produtor

   Produtos fabricados por um produtor são embalados em caixas iguais de dimensões (l, w, h). Estas são colocadas em Paletes de dimensões (L, W, H), onde

H é a altura máxima do carregamento. Em virtude do empacotamento de itens de tamanhos iguais, o PCaP do Produtor tem a tipologia $2/B/O/C$, onde C representa o sortimento de pequenas unidades iguais.

2. PCaP do Distribuidor

   Um distribuidor, como um atacadista por exemplo, geralmente necessita empacotar n caixas de diferentes (l_i, w_i, h_i), $i = 1, ..., n$, em Paletes de tamanho (L, W, H). Este é o chamado PCaP do Distribuidor (PCaPD). Bischoff e Ratcliff [6], e Morabito e Arenales [43] mostraram que esse problema pode ser trabalhado como um Problema de Carregamento de Contêineres (PCC). A tipologia do PCaPD é $2/B/O/$, quando temos apenas um palete para ser usado e este é carregado em camadas (bidimensionais), enquanto a do PCC é $3/B/O/$. No caso de se poder usar vários Paletes, a tipologia do PCaPD é $3/B/I/$.

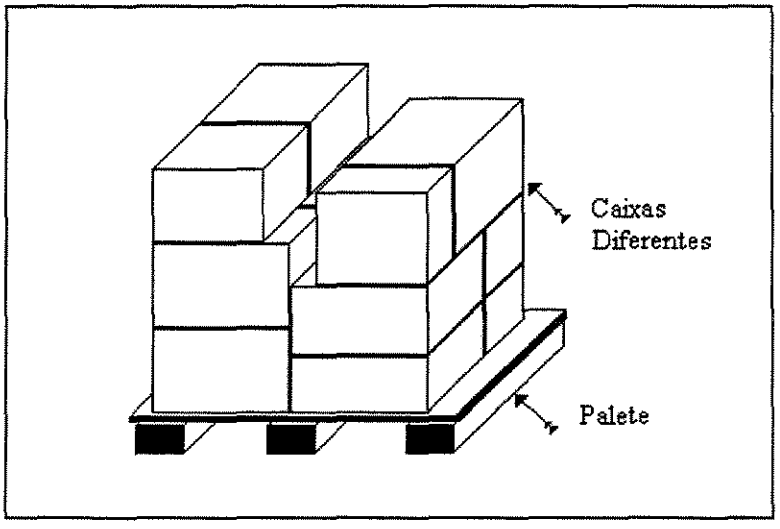


Figura 2.4: Palete do Distribuidor

Capítulo 3

Geração de Colunas num Problema de Corte Unidimensional

3.1 O Método Simplex

O estudo da programação linear tem raízes nos trabalhos de Fourier sobre inequações lineares, publicados em 1826. Kantorovich abordou alguns problemas de programação linear em trabalho de 1939, que foi publicado apenas em 1960 [30] (o mesmo que traz a primeira formulação para o problema de minimizar as perdas num problema de corte de estoque). Mas a programação linear ganhou grande impulso quando Dantzig divulgou em 1947 o Método Simplex, desenvolvido inicialmente para resolver problemas de planejamento da Força Aérea Americana. Em pouco tempo a programação linear foi aplicada à diversas áreas do conhecimento. Koopman, por exemplo, mostrou que a programação linear era muito apropriada para alguns modelos na economia. Em 1975, Kantorovich e Koopman receberam o prêmio Nobel de Economia *"por suas contribuições à teoria de alocação ótima de recursos"*. Aparentemente a Royal Swedish Academy of Sciences, que concede o prêmio, considerou os trabalhos de Dantzig "muito

matemático" para um prêmio de economia.

Antes de abordar o Método Simplex, é importante registrar que nos baseamos em [2], um bom texto introdutório aos Problemas de Corte e Empacotamento, para escrever as duas primeiras seções deste capítulo.

Para estudarmos o Método Simplex, consideremos o seguinte problema de programação linear (PPL):

$$\text{Minimizar} \quad f(x) = \sum_{i=1}^{n} c_i x_i$$

$$\text{sujeito a:} \quad Ax = b$$

$$x \geq 0$$

onde $A \in \mathbb{R}^{m \times n}$ e *posto* $(A) = m$.

Seja $B \in \mathbb{R}^{m \times m}$ uma partição de A tal que *posto* $(B) = m$ e $N \in \mathbb{R}^{m \times n-m}$ tal que $A = (B, N)$. Da mesma forma sejam $x = (x_B, x_N)$ e $c^T = (c_B, c_N)^T$. Chamamos x_B de vetor das variáveis básicas e x_N de vetor das variáveis não básicas.

Temos assim:

$$Ax = b \Leftrightarrow Bx_B + Nx_N = b \Leftrightarrow \ x_B = B^{-1}b - B^{-1}Nx_n$$

Definição . A solução particular do PPL dada por $x_B^0 = B^{-1}b$ e $x_N^0 = 0$ é chamada solução básica . Se $x_B^0 = B^{-1}b \geq 0$ temos uma solução básica primal-viável e dizemos que a partição básica é primal-viável.

Definição . Chamamos o vetor $\pi \in \mathbb{R}^m$ dado por $\pi^T = c_B^T B^{-1}$ de vetor multiplicador simplex ou vetor das variáveis duais. Se $c_j - \pi^T a_j \geq 0$ para $j = 1, ..., n$, então π é uma solução básica dual-viável. (Note que $\pi^T a_j = c_j$ para $a_j \in B$).

Temos o seguinte resultado clássico da Programação Linear:

Teorema 3.1. *Se o PPL têm uma solução ótima então existe uma partição básica ótima.*

Podemos escrever a função objetivo $f(x)$ em termos das variáveis não básicas:

$$\begin{aligned} f(x) &= c_B x_B + c_N x_N = c_B(B^{-1}b - B^{-1}Nx_N) + c_N x_N \\ &= f(x^0) + (c_N - \pi^T N)x_N \\ &= f(x^0) + \sum_{j \in I_N} (c_j - \pi^T a_j)x_j \end{aligned}$$

onde I_N é o conjunto de índices da matriz não básica N.

Desta forma, é fácil ver que a função objetivo, restrita ao sistema $Ax = b$, se altera quando promovemos pertubações na solução básica.

Definição . Chamamos de *Estratégia Simplex* a seguinte pertubação da solução básica:

- escolha $k \in I_N$ tal que $c_k - \pi^T a_k < 0$;
- faça:

$$x_i = \begin{cases} \varepsilon \geq 0 & \text{quando } i = k, \\ 0 & \text{quando } i \in I_N \backslash k \end{cases}$$

A estratégia simplex produz assim uma nova solução dada por:

$$\begin{cases} x_B = x_B^0 + \varepsilon \cdot y \\ x_N = \varepsilon \cdot e_k \end{cases}$$

onde $y = -B^{-1}a_k$ e $e_k = (0, ..., 1, 0, ..., 0)^T$, com 1 na k-ésima componente.

Desta forma o valor da função objetivo é dada por:

$$f(x) = f(x^0) + (c_k - \pi^T a_k)\varepsilon$$

Vale notar que a direção $d \in \mathbb{R}^n$, dada por $d = (d_B, d_N) = (y, e_k)$, define uma pertubação da solução básica e é chamada de direção simplex. Se a solução básica for não degenerada, isto é, $x_b > 0$, segue que d é uma direção de descida viável, visto que, $c^T d = c_k - \pi^T a_k < 0$.

Obviamente desejamos apenas soluções viáveis, o que implica que $x_B \geq 0$. Logo, usamos o seguinte procedimento para obter o maior valor possível de ε:

$$\varepsilon^0 = -\frac{x_{B_i}^0}{y_i} = \text{mínimo} \left\{ -\frac{x_{B_j}^0}{y_j} \;\; y_j < 0, \;\; i = 1, ..., m \right\}$$

onde $x_{B_j}^0$ é a j-ésima componente de x_B^0. Se $y_i \geq 0 \;\; i = 1, ..., m$, o problema é ilimitado.

Com esta escolha de ε, a i-ésima componente de x_B se anula enquanto apenas uma variável não básica se tornou positiva: $x_k = \varepsilon \geq 0$. Obtemos assim uma nova partição básica conforme o seguinte teorema:

Teorema 3.2. *Considere a nova partição $A = (B', N')$, onde a i-ésima coluna de*

B' é dada por $a_k \in N$. A nova partição é básica primal-viável, e sua solução básica associada é dada por $x^1 = x^0 + \varepsilon^0 d$.

Dentre os índices $k \in I_N$ que satisfazem a relação $c_k - \pi^T a_k < 0$, podemos escolher aquele que nos dá a direção simplex com a maior taxa de descida na função objetivo através da seguinte escolha gulosa:

$$c_k - \pi^T a_k = \text{Mínimo } \{c_j - \pi^T a_j \ \ , j = 1, ..., n\}$$

Assim, se $c_k - \pi^T a_k \geq 0$, obtemos uma partição dual-viável e, portanto, uma partição básica ótima.

Desta forma, enquanto a solução básica dual foi inviável, podemos repetir o procedimento acima. Podemos resumir isso no:

`Método Primal-Simplex`

1. Encontre uma partição básica primal-viável: $A = (B, N)$;
2. Faça $x_B = B^{-1}b$;
3. Encontre o menor custo relativo:

   $c_k - \pi^T a_k = \text{Mínimo } \{c_j - \pi^T a_j \ \ , j = 1, ..., n\}$;
4. Se $c_k - \pi^T a_k \geq 0$ $\rightarrow$Solução ótima foi encontrada $\rightarrow$Fim;
5. Determine a direção simplex: $y = -B^{-1}a_k$;
6. Se $y \geq 0$ $\rightarrow$Problema é ilimitado $\rightarrow$FIM;
7. Determine o passo:

   $\varepsilon^0 = -\frac{x^0_{B_i}}{y_i} = \text{mínimo } \{-\frac{x^0_{B_j}}{y_j}, \ \ y_j < 0, \ \ i = 1, ..., m\}$;

8. Atualize a partição básica e volte ao passo 2.

3.2 O Método de Geração de Colunas

Em muitos problemas de otimização linear ou inteira o número de variáveis é muito grande, impossibilitando o uso de métodos exatos como o método primal-simplex. Gilmore e Gomory [21, 22] propuseram, no ínicio da década de 60 do século passado, um método que gera uma coluna após cada iteração do método primal-simplex. Por esse motivo recebeu o nome de Método de Geração de Colunas.

Para melhor compreensão deste método consideremos novamente o PPL:

$$\text{Minimizar} \quad f(x) = \sum_{i=1}^{n} c_i x_i$$

$$\text{sujeito a:} \quad Ax = b$$

$$x \geq 0$$

onde $b \in \mathbb{R}^m$, A é a matriz com todas as n colunas viáveis do problema, X o conjunto de colunas viáveis e $m << n$.

De forma sucinta o Método de Geração de Colunas consiste nos seguintes procedimentos:

1. Encontra-se uma matriz básica B;
2. Gera-se uma coluna $a \in X$ para entrar na base B de forma a melhorar a solução corrente. Caso não seja possível $\rightarrow$FIM;

3. Aplica-se uma iteração do Método Primal-Simplex e volta para 2.

Como vimos na seção anterior, a variável x_k a entrar na base deve ser a correspondente à coluna a_k tal que:

$$c_k - \pi^T a_k = \text{Mínimo } \{c_j - \pi^T a_j \ \ , j = 1, ..., n\} \tag{3.1}$$

Seja $a = (a_1, a_2, ..., a_m)$, $a \in X$, e seu coeficiente na função objetivo $c(a) = \gamma_0 + \sum_{i=1}^{m} \gamma_i a_i$. O custo relativo da variável correspondente à coluna a pode ser determinado por:

$$\varphi(a) = c(a) - \pi^T a = \gamma_0 + \sum_{i=1}^{m} \gamma_i a_i - \sum_{i=1}^{m} \pi_i a_i = \gamma_0 + \sum_{i=1}^{m} (\gamma_i - \pi_i) a_i \tag{3.2}$$

Gilmore e Gomory propuseram, baseado na escolha da variável a entrar na base no método simplex (equação 3.1), encontrar a coluna a que minimiza o custo relativo (equação 3.2):

$$\text{Minimizar} \quad \varphi(a) = \gamma_0 + \sum_{i=1}^{m} (\gamma_i - \pi_i) a_i$$

$$\begin{aligned} \text{sujeito a:} \quad & (a_1, ..., a_m) \in X \\ & a_i \in \mathbb{N}, \qquad\qquad i = 1, ..., m. \end{aligned}$$

Devemos, assim, resolver um sub-problema para obter a coluna a que entrará na base. Detalhadamente temos o seguinte método para resolver o PPL, onde $m << n$:

Método de Geração de Colunas

1. Encontre uma partição básica primal-viável: $A = (B, N)$;

2. Faça $x_B = B^{-1}b$;

3. Encontre o vetor multiplicador simplex $\pi^T = c_B B^{-1}$;

4. Encontre a, solução do seguinte sub-problema:

$$\text{Minimizar} \quad \varphi(a) = \gamma_0 + \sum_{i=1}^{m} (\gamma_i - \pi_i)a_i$$

$$\begin{aligned} \text{sujeito a:} \quad & (a_1, ..., a_m) \in X \\ & a_i \in \mathbb{N}, \qquad\qquad i = 1, ..., m. \end{aligned}$$

5. Se $\varphi(a) \geq 0$ $\rightarrow$Fim;

6. Determine a direção simplex: $y = -B^{-1}a$;

7. Se $y \geq 0$ - Problema é ilimitado $\rightarrow$Fim;

8. Determine o passo: $\varepsilon^0 = -\frac{x^0_{B_i}}{y_i}$ = mínimo $\{-\frac{x^0_{B_j}}{y_j},\ y_j < 0,\ i = 1, ..., m\}$;

9. Atualize a partição básica e volte ao passo 2.

É importante ressaltar que a grande vantagem deste método é a possibilidade de se trabalhar com apenas $m + 1$ colunas explicitamente, durante todo o processo.

3.3 Geração de Colunas num Problema de Corte

Consideremos o seguinte Problema de Corte Unidimensional para Minimizar o número de Objetos Processados (PCOP), como vimos no capítulo 2:

$$\text{Minimizar} \quad c\sum_{j=1}^{n} x_j$$

$$\text{sujeito a:} \quad \sum_{j=1}^{n} a_{ij}x_j = d_i, \quad i = 1, ..., m.$$

$$x_j \in \mathbb{N}, \qquad j = 1, ..., n.$$

As colunas $a_j = (a_{1j}, a_{2j}, ..., a_{mj})$ devem satisfazer a desigualdade $W - \sum_{i=1}^{m} a_{ij}w_i \geq 0$, onde W é a largura da bobina mestre, w_i a largura do item i e cada termo a_{ij} é um inteiro não-negativo, representando a quantidade de itens do tipo i no padrão j, como vimos no segundo capítulo.

Para aplicar o método de Gilmore e Gomory relaxamos a condição de integralidade da solução, ou seja, trabalhamos com variáveis reais não-negativas. Obtemos assim, um problema de programação linear que chamaremos de PCOP Relaxado (PCOPR).

Temos neste caso $\gamma_0 = 1$, considerando o custo de cada objeto processado igual a 1 ($c = 1$), e $\gamma_i = 0$ para $i = 1, ..., m$. Desta forma, para gerarmos uma nova coluna, que corresponde a um novo padrão de corte, temos que resolver o seguinte sub-problema:

$$\text{Minimizar} \quad \varphi(a) = 1 - \sum_{i=1}^{m} \pi_i a_i$$

$$\text{sujeito a:} \quad \sum_{i=1}^{m} w_i a_i \leq W$$

$$a_{ij} \in \mathbb{N}, \qquad i = 1, ..., m.$$

Resolver este sub-problema equivale a obter a solução ótima do seguinte Problema da Mochila (PMI):

$$\text{Maximizar} \quad \sum_{i=1}^{m} \pi_i a_i$$

$$\text{sujeito a:} \quad \sum_{i=1}^{m} w_i a_i \leq W$$

$$a_i \in \mathbb{N}, \qquad i = 1, ..., m.$$

Podemos resolver o Problema da Mochila com algum método exato, como branch-and-bound, que descreveremos na seção 3.5.

3.4 Heurísticas para geração de soluções iniciais

Para aplicar o método de Gilmore e Gomory precisamos, antes de tudo, obter a matriz básica e a respectiva solução inicial. Apresentaremos duas heurísticas simples que se baseiam em estratégias gulosas.

Para gerar m padrões de corte num problema de corte unidimensional, a heurística gulosa mais simples é colocar em cada padrão o maior número possível de um único tipo de item. Vejamos isso mais detalhadamente.

Um padrão de corte com apenas um tipo de item é chamado de padrão de corte homogêneo. Assim, um padrão de corte é homogêneo se e somente se o vetor associado tem apenas uma coordenada não-nula: $a = (0, ..., 0, a_i, 0, ..., 0)$. Num PCOP com m tipos de itens demandados, podemos obter uma matriz para iniciar o processo de geração de colunas com m padrões de cortes homogêneos. Chamaremos de solução inicial homogênea a solução obtida através da seguinte construção:

- $a_i = (0, ..., a_{ii}, ..., 0)$, onde $a_{ii} = \left\lfloor \frac{W}{w_i} \right\rfloor$ $\forall i = 1, ..., m$, ou seja, a_{ii} é o maior inteiro

menor ou igual a $\frac{W}{w_i}$;

- $x_i = \frac{d_i}{a_{ii}}$.

Contudo, podemos construir uma solução de melhor qualidade com uma outra simples heurística gulosa, chamada FFD (First Fit Decreasing). A idéia é alocar preferencialmente os itens maiores, com a maior frequência possível em cada padrão. Assim, inicialmente reordenamos os itens de forma que eles estejam em ordem decrescente em relação à largura. Isto porque, e não é difícil se convencer, os itens maiores são os mais complicados de serem alocados e os itens pequenos, em contra-partida, facilmente são "encaixados" quando não conseguimos mais alocar os itens grandes. A estratégia gulosa é, basicamente, alocar primeiro o maior número possível de itens grandes. Detalhadamente temos a seguinte heurística:

Heurística FFD

1. Seja j=1, $R = \{1, 2, ..., m\}$

2. Faça para $i = 1, ..., m$:

$$b_{ij} = \begin{cases} 0 & \text{se } i \notin R \\ \left\lfloor \frac{W - \sum_{l=1}^{i-1} w_l b_{lj}}{w_i} \right\rfloor & \text{se } i \in R \end{cases}$$

3. $x_j = \frac{d_k}{b_{kj}} = \min_{i \in R}\{d_i / b_{ij} \ , \ \ b_{ij} > 0\}$. (Logo $x_j b_{ij} \leq d_i \ \forall \ i \in R$ e $x_j b_{kj} = d_k$ para algum $k \in R$);

4. $d_i = d_i - x_j b_{ij} \ \forall \ i \in R$;

5. $R = R \backslash k$;

6. $j = j + 1$. Se $j > m$ $\rightarrow$Fim. Caso contrário volte a 2.

Assim, o passo 3 garante que a cada iteração a demanda de um item é atendida exatamente. Após m iterações temos m padrões de corte e uma solução inicial inteira.

Já vimos portanto, como gerar uma solução inicial para o PCOP e, depois, como usar o método de geração de colunas para obter uma solução para o PCOP relaxado. Na próxima seção veremos alguns métodos para arredondar esta solução oriunda do processo de geração de colunas, visto que buscamos uma solução inteira do PCOP, ou seja, quantos objetos (inteiros) serão processados com cada padrão.

3.5 Arredondamento da solução

Trabalharemos nesta seção com o Problema de Corte para Minimizar a Produção com retrições de desigualdade ao invés de restrições de igualdade (PCOP):

$$\text{Minimizar} \quad c\sum_{j=1}^{n} x_j$$

$$\text{sujeito a:} \quad \sum_{j=1}^{n} a_{ij}x_j \geq d_i, \quad i = 1, ..., m.$$

$$x_j \in \mathbb{N}, \qquad j = 1, ..., n.$$

Wascher e Gau [56] dividiram os métodos de arredondamento (para o PCOP) descritos na literatura em três tipos: Aproximação Básica de Padrões (B), Aproximação por Problemas Residuais (R) e Aproximação por Composição de Métodos (C).

Apresentamos nas próximas subseções alguns destes algoritmos, com a denominação proposta por Wascher e Gau. Em todos os casos vamos supor, sem perda de generalidade, que $x = (x_1, ..., x_m)$ é uma solução básica ótima do PCOPR, obtida através do

método de geração de colunas e $x^* = (x_1^*, ..., x_m^*)$ é a solução obtida após o processo de arredondamento.

3.5.1 Aproximação Básica de Padrões

Wascher e Gau apresentam quatro métodos básicos de arredondamento:

- Método BRUSIM

  Este é o mais simples método de arredondamento: para cada variável x_j toma-se o menor inteiro maior ou igual a x_j, ou seja, $x_j^* = \lceil x_j \rceil$.

  As vantagens desse procedimento são óbvias: ele é extremamente rápido e nos dá uma solução viável. Entretanto, tem sido observado que este método sempre resulta em soluções inteiras muito distantes do valor ótimo da função objetivo.

- Método BRURED

  O arredondamento para cima das variáveis de forma simultânea como no Método BRUSIM gera, quase sempre, um excesso de produção que pode ser minorado. De fato, podemos checar, após a aplicação deste método, quais variáveis podem ser diminuídas em uma unidade sem deixar de atender a demanda. Assim, verificamos, para cada $k \in \{1, 2, ..., n\}$, se a variável x_k, após o método BRUSIM, satisfaz a desigualdade:

$$a_{ik}(x_k - 1) + \sum_{j=1; j \neq k}^{m} a_{ij} x_j \geq d_i \ , \ i = 1, ..., m$$

  Se x_k satisfaz tal desigualdade fazemos $x_k^* = x_k - 1$.

- Método BRUSUC

  Neste caso fixamos todas as variáveis x_k inteiras e também a variável com maior parte fracionária é arredondada para o inteiro superior, isto é, se x_k é inteira

fazemos $x_k^* = x_k$ e $x_l^* = \lceil x_l \rceil$, onde $\lceil x_l \rceil - x_l \geq \lceil x_i \rceil - x_i$, $\forall\, i = 1, ..., m$. Obtemos assim um outro problema de programação linear:

$$\text{Mininimizar} \quad \sum_{j \in J_B} \bar{x}_j$$

$$\text{sujeito a:} \quad \sum_{j \in J_B} a_{ij} \bar{x}_j \geq d_i, \quad i = 1, ..., m$$

$$\bar{x}_j \geq 0, \qquad j \in J_B$$

onde:

J_B é o conjunto de índices das variáveis básicas;

$\bar{x}_j = x_j^*$, $j \in J_B$ se x^* é inteiro;

a variável x_l com a maior parte fracionária é arredondada para cima: $\bar{x}_l = \lceil x_l \rceil$.

Repete-se, então, este processo até todas as variáveis x_k^*, $k = 1, ..., m$, serem inteiras. Isto ocorre em no máximo m iterações, pois em cada iteração pelo menos uma variável é fixada.

- Método BOPT

  A primeira iteração deste método em nada difere do Método BRUSUC: fixa-se todas as variáveis x_k inteiras e também a variável com maior parte fracionária é arredondada para o inteiro superior. Em seguida, resolve-se o problema resultante através de um método de Programação Inteira.

3.5.2 Aproximação por Problemas Residuais

O primeiro passo nestes métodos é arredondar para baixo a solução ótima do PCOPR, ou seja, trabalha-se com o vetor de frequências inteiras $\overline{x} = (\lfloor x_1 \rfloor, ..., \lfloor x_m \rfloor)$.

Como estamos assumindo que a solução do PCOPR não é inteira, $\overline{x}$ não é uma solução viável, ou seja, após o processo de arredondamento para baixo alguns itens não terão as demandas atendidas. Definimos, assim, a demanda residual como sendo:

$$dr_i = \max\left(0, d_i - \sum_{j=1}^{n} a_{ij}\lfloor x_j \rfloor\right), \quad i = 1, 2, ..., m.$$

Com isto, temos o seguinte Problema Residual (PR):

$$\begin{aligned} \text{Minimizar} \quad & \sum_{j=1}^{n} x_j \\ \text{sujeito a:} \quad & \sum_{j=1}^{n} a_{ij}x_j \geq dr_i, \quad i = 1, ..., m \\ & x_j \in \mathbb{N}, \qquad\qquad j = 1, ..., n. \end{aligned}$$

Os chamados algoritmos de aproximação por Problemas Residuais se diferenciam pelo método utilizado para resolver o PR.

Vale ressaltar que, uma vez que as demandas residuais são bem menores que as demandas do problema original, o Problema Residual é geralmente tratado como um Problema de Empacotamento de *Bin's*.

- Método ROPT

  O problema residual é resolvido por um método de Programação Inteira. Assim, obtém-se uma solução ótima do PR que, composta com a solução $\overline{x}$ obtida com o arredondamento para baixo da solução do PCOPR, não necessariamente é solução ótima do PCOP. Caso o número m de itens seja grande, é preciso resolver um Problema Residual Grande, o que pode tornar este procedimento proibitivo

computacionalmente, visto que o Problema de Empacotamento de *Bin's* é $\mathcal{NP}$-*difícil no sentido forte* (ver apêndice A).

- Método RFFD

  O problema residual é resolvido usando-se a heurística FFD. Com isto, este método é bem mais rápido computacionalmente em relação ao ROPT.

- Método RSUC

  Neste método, o Problema Residual também é relaxado em relação às condições de integralidade e resolvido através do Método de Geração de Colunas. Este processo é repetido até o arredondamento para baixo resultar em frequências nulas. No caso de alguma demanda não ser atendida, resolve-se o problema residual corrente através de um algoritmo exato para o Problema de Empacotamento de *Bin's*.

3.5.3 Aproximação por Composição de Métodos

Stadler [46] apresentou um procedimento que combina idéias da Aproximação Básica de Padrões e da Aproximação por Problemas Residuais. Ele inicia com o método BRUSUC para gerar uma solução viável para o PCOP. Como neste estágio geralmente ocorre excesso de produção, tenta-se reduzi-lo iterativamente: identifica-se o padrão que contém o maior número de itens com excesso de produção; reduz-se a frequência deste padrão em uma unidade. Este processo é repetido até não existir mais excesso de produção. Obtemos assim, um Problema Residual que pode ser resolvido por um método exato ou heurístico. Temos neste caso dois procedimentos possíveis:

- Método CSTAOPT

  Inicialmente o método BRUSUC é aplicado, seguido do procedimento de redução de Stadler, como descrevemos acima. O problema residual é resolvido através de

um algoritmo exato.

- Método CSTAFFD

  O que difere este método do CSTAOPT é a resolução do Problema Residual, que neste caso se dá com a aplicação da heurística FFD.

3.5.4 Desempenho dos métodos

Wascher e Gau [56] realizaram testes para aferir o comportamento e qualidade de cada um destes métodos. Ao todo, aplicaram os métodos para 5 classes de problemas, com diversas quantidades de itens diferentes, $m = 10$, $m = 20$, $m = 30$, $m = 40$, $m = 50$. Em cada classe foram resolvidos 800 problemas. Dentre os resultados apresentados destacamos os da tabela 3.1.

Método	# de Problemas onde a solução ótima foi obtida	Tempo Total [s]
BRUSIM	188	14538
BRURED	821	14538
BRUSUC	578	17343
BOPT	1472	282102
ROPT	3874	18887
RFFD	3701	14538
RSUC	3921	18470
CSTAOPT	3709	17859
CSTAFFD	3630	17625

Tabela 3.1: Qualidade da solução e tempo computacional dos Métodos de Aproximação

Observação. Wascher e Gau usaram um PC-486, 66 MHz nos testes com exceção do Método BOPT. Também com exceção ao BOPT, o tempo computacional se refere ao tempo usado na resolução do PCOPR usando o Método de Geração de colunas mais o

tempo do método de arredondamento. O tempo atribuído ao Método BOPT se refere apenas ao tempo computacional do método heurístico, e foi obtido numa estação de trabalho HP9000/720, 57 MIPS, visto que o método se mostrou muito extenso em relação ao tempo computacional.

Em relação aos métodos de Aproximação Básica de Padrões é possível notar que o BOPT é superior em relação a qualidade da solução. Porém, é muito caro computacionalmente. Já o BRURED é rápido computacionalmente e tem soluções melhores em geral do que o BRUSUC e o BRUSIM. Por conseguinte, o BRURED é o método mais competitivo entre os métodos desta classe.

É notória também a superiodade dos Métodos de Aproximação por Problemas Residuais se comparados com os de Aproximação Básica de Padrões em relação a qualidade da solução. O método RFFD é o mais rápido desta classe de métodos e atingiu a solução ótima em $92,5\%$ dos problemas. Já o RSUC atingiu a solução ótima em 98% dos problemas. Vale destacar que o CSTAOPT, que obteve o ótimo em $92,5\%$, e o RSUC ficaram no máximo uma unidade da solução ótima em todos os problemas testados. Já ROPT, que obteve o ótimo em $96,85\%$ dos problemas testados, não apresentou esta propriedade de ficar no máximo uma unidade do ótimo em todos os problemas. Desta forma, CSTAOPT e RSUC são, destacadamente, os métodos de arredondamento mais competitivos tratados pelos autores do referido artigo.

Poldi apresentou em [45] algumas úteis modificações aos métodos clássicos de arredondamento, obtendo resultados competitivos.

Por fim, é importante ressaltar que, no caso de trabalharmos com o Problema de Corte para Minimizar o número de Objetos Processados e o Setup (PCOPS), os métodos BRUSIM, BRURED, BRUSUC e BOPT apresentam a vantagem de não aumentar o número de padrões distintos. Voltaremos a este assunto no capítulo 6.

3.6 Resolvendo o Problema da Mochila

Martello e Toth [38] é uma indispensável referência para o estudo do Problema da Mochila em suas diversas variações. Extraímos desse importante livro uma brevíssima apresentação do que consideramos ser as idéias centrais dos métodos de resolução do Problema da Mochila 0-1 (PM0-1), do Problema da Mochila Ilimitado (PMI) e do Problema da Mochila Limitado (PML).

Comecemos por descrever o limitante superior para o PM0-1 obtido por Dantzig em 1957. Para isso, relaxamos inicialmente as condições de integralidade do PM0-1, obtendo o problema de programação linear PM0-1R:

$$\text{Maximizar} \quad \sum_{i=1}^{n} u_i x_i$$

$$\text{sujeito a:} \quad \sum_{i=1}^{n} w_i x_i \leq W$$

$$0 \leq x_i \leq 1, \qquad i = 1, ..., n.$$

Assumimos sem perda de generalidade que:

$$u_j \ , \ \ w_j \ \text{ e } \ W \ \text{ são inteiros positivos}$$

$$\sum_{j=1}^{n} w_j > W$$

$$w_j \leq W, \ \ j = 1, 2, ..., n.$$

Sem perda de generalidade podemos supor também que os itens estão em ordem decrescente em relação a densidade (utilidade/peso), isto é:

$$\frac{u_1}{w_1} \geq \frac{u_2}{w_2} \geq ... \geq \frac{u_n}{w_n}$$

Suponhamos que os itens são consecutivamente inseridos na mochila até o primeiro item s não poder ser colocado por exceder a capacidade. Este item é chamado de item crítico e é dado, mais especificamente, por:

$$s = \min \left\{ j : \sum_{i=1}^{j} w_i > W \right\}$$

Devido as hipóteses assumidas, temos que $1 < s \leq n$. Dantzig apresentou em 1957 o seguinte teorema:

Teorema 3.3. *A solução ótima $\bar{x}$ de PM0-1R é:*

$$\begin{aligned} \bar{x}_j &= 1 \ \textit{para} \ j = 1, ..., s-1 \\ \bar{x}_j &= 0 \ \textit{para} \ j = s+1, ..., m \\ \bar{x}_s &= \frac{\bar{W}}{w_s} \end{aligned}$$

onde

$$\bar{W} = W - \sum_{j=1}^{s-1} w_j$$

Demonstração: É fácil ver que qualquer solução x de PM0-1R deve ser maximal, no sentido de $\sum_{j=1}^{n} w_j x_j = W$. Sem perda de generalidade, podemos supor que $u_j/w_j > u_{j+1}/w_{j+1}$ para todo $j = 1, ..., n$. Seja x^* solução ótima do PM0-1. Suponhamos por absurdo que $x_k^* < 1$ para algum $k < s$. Devemos ter, assim, $x_q^* > \bar{x}_q$ para algum

$q \geq s$. Dado um $\varepsilon > 0$ suficientemente pequeno, podemos aumentar o valor de x_k^* por ε decrescendo o valor de x_q^* por $\varepsilon w_k/w_q$. O valor da função objetivo aumenta, assim, em $\varepsilon(u_k - u_q w_k/w_q) > 0$, visto que $u_k/w_k > u_q/w_q$. Isto contradiz a hipótese de otimalidade da solução. Da mesma forma mostramos que $x_k^* > 0$ para $k > s$ é impossível. Portanto, como toda solução ótima é maximal, temos $\bar{x}_s = \bar{W}/w_s$.

Assim, o valor ótimo do PM0-1R é dada por:

$$z(\text{PM0-1R}) = \sum_{j=1}^{s-1} u_j + \bar{W}\frac{u_s}{w_s}$$

Devido a integralidade de u_j e x_j, temos o seguinte limitante superior para o PM0-1:

$$L_1 = \lfloor z(\text{PM0-1R}) \rfloor = \sum_{j=1}^{s-1} u_j + \left\lfloor \bar{W}\frac{u_s}{w_s} \right\rfloor$$

Martello e Toth em [36] obtiveram o primeiro limitante que domina o limitante de Dantzig:

Teorema 3.4. *Seja*

$$L^0 = \sum_{j=1}^{s-1} u_j + \left\lfloor \bar{W}\frac{u_{s+1}}{w_{s+1}} \right\rfloor$$

$$L^1 = \sum_{j=1}^{s-1} u_j + \left\lfloor u_s - (w_s - \bar{W})\frac{u_{s-1}}{w_{s-1}} \right\rfloor$$

Então:

1. *Um limitante superior de PM0-1 é*

$$L_2 = \max\{L^0, L^1\}$$

2. *Para qualquer PM0-1 temos* $L_2 \leq L_1$.

A demonstração pode ser encontrada também em [38].

Em relação aos métodos de resolução, nos concentraremos principalmente no chamado método *branch-and-bound.* Segundo Martello e Toth [38] o primeiro algoritmo *branch-and-bound* capaz de obter a solução exata de um Problema da Mochila 0 ou 1 (PM0-1) foi apresentado por Kolesar em 1967 [31]. Este consistia em um método de busca binária: (a) em cada nó, seleciona um item ainda não fixado j tendo máximo lucro por unidade de peso (densidade) e gera dois nós descendentes, fixando x_j em 1 e 0; (b) continua a busca por um nó viável cujo valor do limite superior L_1 é máximo.

Horowitz e Sahni [28] apresentaram em 1974 um algoritmo *branch-and-bound* com algumas modificações em relação a proposta de Kolesar. Definiram basicamente dois tipos de movimento:

- **Movimento para frente (*Forward move*)**: Inserção da maior quantidade possível (inteira) de itens consecutivos na solução corrente.
- **Movimento para trás (*Backtracking move*)**: Remoção do último item inserido na solução corrente.

Quando um *forward move* é esgotado, o limite superior L_1 correspondente à solução corrente é calculado e comparado com a melhor solução encontrada. Se não for possível melhorar a solução faz-se um *backtracking move.* Caso contrário, faz-se um *forward move.* Se o último item tiver sido considerado, compara-se a solução corrente com a melhor solução encontrada para uma possível atualização. Se não for possível fazer mais nenhum *backtracking move* o algoritmo termina.

Martello e Toth [36] apresentaram em 1977 algumas modificações a esta proposta de Horowitz e Sahni, obtendo um método mais eficiente. Entre estas mudanças está a troca do limitante superior usado na comparação, L_2 ao invés do L_1. Os mesmos Martello e Toth, em outro artigo publicado no mesmo ano [37], adaptaram este algoritmo para resolver o Problema da Mochila Ilimitado.

Transcrevemos no apêndice B o pseudo-código de um *branch-and-bound* sugerido por Chvatal [8] para gerar uma coluna, solução de um PMI, para o problema de corte unidimensional.

No mesmo artigo que trouxe um algoritmo *branch-and-bound* para o PMI, Martelo e Toth [37] apresentaram um *branch-and-bound* para o Problema da Mochila Limitada (PML), transformado este num PM0-1.

Consideremos o seguinte PML:

$$\begin{aligned} \text{Maximizar} \quad & \sum_{i=1}^{n} u_i x_i \\ \text{sujeito a:} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \leq b_i, \qquad i = 1, ..., n \\ & x_i \in \mathbb{N} \qquad i = 1, ..., n. \end{aligned}$$

onde b_i é o número máximo de objetos do tipo i que podem ser postos na mochila.

O algoritmo *TB01*, descrito também em [38], transforma o PML acima num PM0-1 com $\bar{n}$ variáveis, com utilidades $\bar{u}_i$, peso $\bar{w}_i$ para todo $i = 1, ..., \bar{n}$ e capacidade da mochila $\bar{W}$:

Pseudo-Código TB01 - Transforma PML em PM01

Início

$\bar{n} = 0$

para j=(1) até n

Início

$\beta = 0$

$k = 1$

repita

se $\beta + k > b_j$ então $k = b_j - \beta$

$\bar{n} = \bar{n} + 1$

$\bar{un} = ku_j$

$\bar{wn} = kw_j$

$\beta = \beta + k$

$k = 2k$

até $\beta = b_j$

Fim

Fim

Vale notar que o problema transformado tem $\bar{n} = \sum_{j=1}^{n} \lceil \log_2(b_j + 1) \rceil$ variáveis.

Em nossa implementação, que detalharemos no capítulo 6, utilizamos o algoritmo *MTB2*, disponibilizado por Martello e Toth em [38], implementado na linguagem Fortran. Este algoritmo utiliza o *TB01* para transformar o PML em um PM0-1. Neste PM0-1 aplica-se um algoritmo, *MT2* [38], que contém o método *branch-and-bound* por eles desenvolvido.

Capítulo 4

Minimização do Setup num Problema de Corte Unidimensional

Neste capítulo estudaremos alguns métodos de resolução do Problema de Corte unidimensional para minimizar o número de Objetos Processados e o Setup (PCOPS):

$$\begin{aligned} &\text{Minimizar} && c_1 \sum_{j=1}^{n} x_i + c_2 \sum_{j=1}^{n} \delta(x_j) && \\ &\text{sujeito a:} && \sum_{j=1}^{n} a_{ij} x_j = d_i, && i = 1, ..., m. \\ & && x_j \in \mathbb{N}, && j = 1, ..., n. \end{aligned}$$

onde:

c_2 - custo de *setup*;

$$\delta(x_j) = \begin{cases} 1 & \text{se } x_i > 0, \\ 0 & \text{se } x_i = 0. \end{cases}$$

Este problema apresenta dois objetivos conflitantes: minimizar o número de Objetos Processados bem como o número de padrões distintos. Como dissemos no segundo capítulo, este problema é de difícil resolução e tem grande importância no planejamento da produção de algumas indústrias.

A dificuldade em se resolver este problema resulta da descontinuidade da função objetivo. De fato, não podemos relaxar as condições de integralidade da solução e usar o Método de Geração de Colunas, pois não temos linearidade e continuidade na função objetivo. Não por acaso a grande maioria dos métodos publicados para resolver o PCOPS são heurísticos e não abordam esta formulação.

Não nos propomos a fazer um estado-da-arte neste capítulo. Apresentamos quatro métodos diferentes que representam quatro diferentes abordagens, presentes na literatura, para o mesmo problema: um procedimento heurístico sequencial (SHP); um procedimento de redução de padrões por combinação, também chamado de procedimento pós-otimização (Kombi); um método exato, *branch-cut-price*; um método que reduz o desvio quadrático da produção em relação à demanda para um setup fixo (ILS-APG).

4.1 Heurística SHP

Este método foi proposto por Haessler [23] e se baseia numa técnica exaustiva de repetição de padrão (Sequential Heuristic Procedure - SHP): em cada iteração calcula-se alguns parâmetros de aspiração, faz-se uma busca por padrões de corte que satisfaçam tais parâmetros até que todas as demandas sejam atendidas.

Antes de descrever detalhadamente o método vamos explicitar os parâmetros de aspiração utilizados em cada iteração :

- MAXTL: máximo de desperdício tolerável;
- MAXR: número máximo de itens permitidos no padrão - depende do número de facas da máquina de corte;
- MINR: número mínimo de itens permitidos no padrão;
- MINU: número mínimo de objetos processados com o padrão.

Em cada iteração do método SHP calcula-se a demanda residual dr_i para cada item i. Isto porque na j-ésima iteração se define não só o padrão $a_j = (a_{1j}, a_{2j}, ..., a_{mj})$ como também o valor da variável x_j. Definimos assim:

$$dr_i = \max \ \{0, d_i - \sum_{k=1}^{j} a_{ik}x_k\} \ , \quad i = 1, 2, ..., m.$$

Obviamente, antes do processo se iniciar temos $dr_i = d_i$, $i = 1, 2, ..., m$.

Em cada iteração duas estimativas são fundamentais:

- $NR = \frac{\sum_i dr_i w_i}{W}$ - número de objetos que ainda devem ser processados para satisfazer a demanda;
- $\overline{NI} = \frac{\sum_i dr_i}{NR}$ - número médio de itens obtidos em cada bobina.

No referido artigo, Haessler propôs também os valores a serem atribuídos a cada parâmetro de aspiração :

- MAXTL $\in [0.006W, 0.03W]$;
- MAXR: igual a capacidade máxima da máquina de corte;
- MINR $= \overline{NI} - 1$;
- MINU= $\alpha \cdot NR$ onde $\alpha \in [0.5, 0.9]$.

Observação: Os parâmetros MAXTL e MINU introduzem novas restrições ao padrão que deve ser gerado na iteração j: $W - \sum_{i=1}^{m} a_{ij} w_i \leq MAXTL$, $x_j \geq MINU$.

Podemos agora apresentar a heurística SHP:

Heurística SHP

1. Dados de entrada: W, m, w_i e d_i para cada $i = 1, ..., m$, MAXTL, MAXR, α;
2. Calcula NR, $\overline{NI}$, MINR, MINU;
3. Calcula $b_i = \lfloor \frac{d_i}{MINU} \rfloor$, $i = 1, ..., m$ (limitante superior para cada item no padrão a ser construído);
4. Ordena os itens por ordem decrescente de demanda residual;
5. Gera padrões de corte em ordem lexicográfica, respeitando os limitantes b_i, garantindo que o item com maior demanda residual esteja no padrão, até que se tenha um padrão que satisfaça os três parâmetros de aspiração (MAXTL, MAXR, MINR);
6. Se a busca por um padrão satisfazendo os critérios de aspiração falhou, vá para o passo 8, caso contrário, tendo obtido um padrão de corte $a_j = (a_{1j}, a_{2j}, ..., a_{mj})$, faça $x_j = \min \{\lfloor d_i / a_{ij} \rfloor \ \ i = 1, ..., m\}$;
7. Atualiza d_i. Se $d_i = 0 \ \forall \ \ i = 1, ..., m \rightarrow$FIM, caso contrário volte a 2;

8. Se $MINU > 1$ faça $MINU = MINU - 1$ e volte a 3;

9. Se $MINU = 1$ faça a busca descrita em 5, selecionando o padrão com o menor desperdício e volte a 6;

Observação: O passo 8 amplia o conjunto de possíveis padrões de corte a serem gerados. Já o passo 9 garante que o processo termina num número finito de passos e com uma solução viável.

Vale notar que o método não usa os valores de c_1 e c_2, apesar da formulação da função objetivo do PCOPS ter sido proposta no referido artigo. No caso de se necessitar diminuir o número de padrões distintos (setup), deve-se permitir padrões de corte com maior desperdício, ou seja, aumentar o parâmetro MAXTL.

O SHP é um método que nos dá uma boa solução para o PCOPS e não é caro computacionalmente. Por este motivo muitos métodos publicados na literatura utilizam o SHP para gerar uma solução inicial e também como base de comparação para aferir a qualidade dos mesmos. No próximo capítulo apresentaremos algumas modificações que fizemos no algoritmo original que descrevemos acima.

4.2 Heurística KOMBI

Este método foi desenvolvido por Foester e Wascher [18] e se baseia na combinação de padrões visando diminuir o setup de uma solução inicial. Este processo de reduzir o número de padrões num procedimento pós-otimização foi mencionado inicialmente por Hardley [25]. Outros métodos baseados nesta idéia foram publicados por Johnston [33], Allwood e Goulimis [1] e Diegel et al. [11]. Todos têm em comum a combinação de pares ou triplas de padrões, mas diferem na forma em que essas combinações são realizadas.

O método Kombi pode ser visto como uma generalização do método de Diegel et al. [11]. O Kombi se baseia no simples fato da soma das frequências dos padrões resultantes ser igual a soma das frequências dos padrões originais, ou seja, mantém constante o número de objetos processados com uma possível redução do setup.

Mais detalhadamente vamos descrever a combinação 2 para 1 proposta por Diegel et al.. Sejam dois padrões 1 e 2 com as respectivas frequências x_1 e x_2. Para cada item de largura w_i, $i = 1, ..., m$, a soma $s_i = a_{i1}x_1 + a_{i2}x_2$ representa o número de itens do tipo i fornecidos pelos dois padrões, enquanto $x_* = x_1 + x_2$ é o número total de bobinas processadas com os padrões 1 e 2. Estes dois padrões podem ser substituídos por um outro padrão $a_* = (a_{1*}, ..., a_{m*})$, com $a_{i*} = s_i/x_*$, $i = 1, ..., m$ se todas as razões s_i/x_* forem inteiras. Assim, utilizando o novo padrão x_* vezes, teremos a mesma produção de cada item i, $i = 1, ..., m$.

Como dissemos, o KOMBI generaliza este método, baseando-se no fato da soma das frequências dos novos padrões serem iguais à soma das frequências dos padrões iniciais. Vejamos três tipos de procedimentos:

1. Combinações 3 para 2 e 3 para 1

   Baseado nesta propriedade, podemos obter dois padrões $*$ e $**$, com frequências x_* e x_{**}, de três padrões, 1, 2 e 3, de pelo menos três maneiras diferentes (combinação 3 para 2):

   $$\begin{aligned} x_* &= x_1 \quad \text{e} \quad x_{**} = x_2 + x_3; \\ x_* &= x_2 \quad \text{e} \quad x_{**} = x_1 + x_3; \\ x_* &= x_3 \quad \text{e} \quad x_{**} = x_1 + x_2. \end{aligned}$$

   Para obter os dois padrões $* = (a_{1*}, ..., a_{m*})$ e $** = (a_{1**}, ..., a_{m**})$, adota-se o seguinte procedimento:

   - Calcula as somas $s_i = a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3$, $i = 1, ..., m$;

- Determina, recursivamente, todas as possíveis decomposições de s_i, isto é, todos os pares (a_{i*}, a_{i**}), a_{i*}, a_{i**} inteiros, tal que $s_i = a_{i*}x_* + a_{i**}x_{**}$, onde os padrões $*$ e $**$ são viáveis (se a viablidade de algum padrão é violada o procedimento recursivo é abortado).

Da mesma forma podemos realizar a combinação 3 para 1.

2. Combinações 4 para 3, 4 para 2 e 4 para 1

   Consideremos 4 padrões arbitrários 1, 2, 3 e 4 que serão combinados em 3 padrões $*$, $**$ e $***$ com as seguintes frequências:

$$x_* = x_{j_1} \ , \ \ x_{**} = x_{j_2} \ , \ \ x_{***} = x_{j_3} + x_{j_4}$$

   onde $j_1, j_2, j_3, j_4 \in \{1, 2, 3, 4\}$ são dois a dois distintos.

   Podemos ter seis tipos de combinações diferentes. Analogamente, deve-se determinar todas as combinações possíveis e viáveis $(a_{i*}, a_{i**}, a_{i***})$ de s_i, satisfazendo $s_i = a_{i*}x_* + a_{i**}x_{**} + a_{i***}x_{***}$. Vale ressaltar que as combinações 3 para 2 e 2 para 1 são casos particulares da combinação 4 para 3.

   Na combinação de 4 padrões em 2 (4 para 2) temos 7 conjuntos de frequências x_* e x_{**} para os padrões resultantes $*$ e $**$:

$$x_* = x_{j_1} \ \text{ e } \ x_{**} = x_{j_2} + x_{j_3} + x_{j_4}$$

   ou

$$x_* = x_{j_1} + x_{j_2} \ \text{ e } \ x_{**} = x_{j_3} + x_{j_4}$$

   onde $j_1, j_2, j_3, j_4 \in \{1, 2, 3, 4\}$ são dois a dois distintos. O mesmo procedimento recursivo identifica os dois padrões $*$ e $**$. Da mesma forma, desenvolvemos a combinação 4 para 1.

Estes procedimentos podem ser generalizados para combinações p para q, com $p > q$ quaisquer. Porém, para não ser muito caro computacionalmente o método KOMBI se limitou a trabalhar com, no máximo, 4 padrões e com reduções p para (p-1). A implementação mais extensa deste método recebeu o nome de KOMBI234:

`Pseudo-Código do KOMBI234`

Enquanto combinações 2 para 1 são encontradas faça:

Para todos os pares de padrões j_1, j_2 faça:

combinação 2 para 1

Fim

Fim

Enquanto combinações 3 para 2 são encontradas faça:

Para todos as triplas de padrões j_1, j_2, j_3 faça:

combinação 3 para 2

Fim

Fim

Enquanto combinações 4 para 3 são encontradas faça:

Para todos as quádruplas de padrões j_1, j_2, j_3, j_4 faça:

combinação 4 para 3

Fim

Fim

Para reduzir ainda mais o tempo computacional pode se limitar o KOMBI às combinações 2 para 1 e 3 para 2. Esta implementação do Kombi recebe o nome de KOMBI23.

Segundo os resultados apresentados por Foester e Wascher [18], KOMBI reduziu o setup em mais de 60% dos problemas testados. Apresentou ainda melhores resultados, em quase todos os casos, em relação aos métodos similares e também se mostrou superior ao SHP, descrito na seção anterior. Apresentaremos alguns resultados do KOMBI234 no capítulo 6. Estes servirão de parâmetro para aferir a qualidade do método que estamos propondo.

Limeira apresentou em [34] uma variante no método Kombi, além de outras três heurísticas que buscam obter um número reduzido de padrões distintos.

4.3 Método exato *branch-cut-price*

Vanderbeck [52] propôs modelar o problema de minimização do setup como de Programação Inteira não-linear, e chamou este de Problema de Minimização de Padrão (PMP). Inicialmente, o PMP tem a seguinte formulação chamada P:

$$Z = \text{Minimizar} \sum_{k=1}^{K} y_k$$

sujeito a:

$$\sum_{k=1}^{K} z_k x_{ik} = d_i \quad i = 1, ..., n \tag{4.1}$$

$$\sum_{k=1}^{K} z_k \leq K \tag{4.2}$$

$$z_k \leq K y_k \quad k = 1, ..., K \tag{4.3}$$

$$\sum_{i=1}^{m} w_i x_{ik} \leq W y_k \quad k = 1, ..., k \tag{4.4}$$

$$x_{ik} \in \mathbb{N} \quad i = 1, ..., m\ , \quad k = 1, ..., K \tag{4.5}$$

$$y_k \in \{0, 1\} \quad k = 1, ..., K \tag{4.6}$$

$$z_k \in \mathbb{N} \quad k = 1, ..., K \tag{4.7}$$

onde:

- K é número de bobinas em estoque que podem ser processadas (obtido resolvendo-se o PCOP padrão, conforme definimos no segundo capítulo);
- z_k é o número de vezes que o padrão k é usado;
- $y_k = 1$ se o padrão k é usado e 0 caso contrário;
- x_{ik} é o número de itens do tipo i no padrão k.

O objetivo é, assim, minimizar o número de diferentes padrões de corte que são utilizados. As restrições 4.1, que são não-lineares, asseguram que as demandas são atendidas exatamente. A restrição 4.2 corresponde ao limitante (K) do número de bobinas a serem utilizadas. A restrição 4.3 se refere à propriedade da variável y_k. Por fim, a restrição 4.4 garante a viabilidade dos padrões de corte.

Visto que a formulação P é um problema de programação inteira não-linear, trabalha-se com uma relaxação fraca, resultando num problema de programação inteira

linear. Para isso, usa-se a decomposição de Dantizg-Wolfe adaptada para programação inteira [51]. Obtém-se inicialmente K sub-problemas idênticos, que consistem em selecionar um padrão de corte viável e fixar o número de vezes que ele é utilizado. A formulação mestre (M) abaixo é uma reformulação do PMP em termos das soluções desses sub-problemas:

$$Z^M = \text{Minimizar} \quad \sum_{q \in Q} \lambda_q \tag{4.8}$$

$$\text{sujeito a:} \quad \sum_{q \in Q} q_0 q_i \lambda_q = d_i \quad i = 1, ..., m \tag{4.9}$$

$$\sum_{q \in Q} q_0 \lambda_q \leq K \tag{4.10}$$

$$\lambda_k \in \{0, 1\}, \quad q \in Q \tag{4.11}$$

onde:

- $Q = \{q = (q_0, q_1, ..., q_m) \in \mathbb{N}^{m+1} : \sum_{i=1}^{n} w_i q_i \leq W \ , \ q_0 q_i \leq d_i \ \forall \ i\}$, é o conjunto das soluções viáveis para as restrições (4.3–4.7) para um k fixo e uma relaxação da restrição 4.1.

- λ_q é uma variável associada a cada ponto $q \in Q$, que assume valor 1 se o padrão de corte $(q_1, ..., q_n)$ é usado q_0 vezes, e assume valor 0 caso contrário.

Vale observar que $\lambda_q = 1$ na formulação $[M]$ é equivalente a $(x_{1k}, ..., x_{nk}, y_k, z_k) = (q_1, ..., q_n, 1, q_0)$ para algum k na formulação $[P]$.

A formulação $[M]$ é um problema de programação inteira linear com grande número de variáveis. As não linearidades de $[P]$ estão implícitas nas definições das colunas. Devido ao grande número de colunas e variáveis associadas em $[M]$, utiliza-se um

procedimento de geração de colunas num problema de programação inteira, chamado algoritmo *branch-and-price* (Barnhart e outros [4], Vanderbeck e Wolsey [54]). Assim, este método consiste em inserir o método de geração de colunas num algoritmo *branch-and-bound*. Em cada nó da árvore do método *branch-and-bound*, é obtido um limitante inferior resolvendo-se uma relaxação linear do problema mestre corrente.

Este método resolve exatamente problemas de pequeno porte. Porém, falha na obtenção de soluções ótimas em problemas de tamanhos moderados e grandes.

4.4 Heurística ILS-APG

Umetami, Yagiura e Ibaraki [49] desenvolveram o "Iterated Local Search algorithm with Adaptive Pattern Generations" (ILS-APG) cuja idéia básica é, fixando o número de padrões distintos (*setup*), buscar uma solução com desvio quadrático, dos itens produzidos em relação à demanda, suficientemente pequeno.

Para descrever os modelos apresentados por Umetami, Yagiura e Ibaraki precisamos definir algumas novas notações: seja S o conjunto de todos os padrões viáveis, $\Pi \subseteq S$ o conjunto solução do PCOPS e $x = (x_1, x_2, ..., x_{|\Pi|}) \in \mathbb{Z}_+^{|\Pi|}$, onde x_j representa a frequência do padrão $p_j \in \Pi$ e $|\Pi|$ é igual ao número de elementos do conjunto Π, ou seja, o número de padrões distintos.

Podemos agora apresentar o MIN-PAT:

$$\text{Minimizar} \quad |\Pi|$$

$$\text{sujeito a:} \quad \sum_{i=1}^{m} \left(\sum_{p_j \in \Pi} a_{ij} x_j - d_i \right)^2 \leq D$$

$$\Pi \subseteq S$$

$$x_j \in \mathbb{N}, \qquad \text{para } p_j \in \Pi$$

onde D é o limitante do desvio quadrático das demandas.

Vale notar que, se $D = 0$, MIN-PAT é o mesmo problema de minimização de Padrões (PMP) que abordamos na seção anterior. Mas como dissemos, o ILS-APG fixa o *setup* e busca minimizar o desvio quadrático em relação à demanda conforme o seguinte modelo, chamado de MIN-DEV:

$$\text{Minimizar} \quad f(\Pi, x) = \sum_{i=1}^{m} \left(\sum_{p_j \in \Pi} a_{ij} x_j - d_i \right)^2$$

$$\text{sujeito a:} \quad \Pi = n$$

$$\Pi \subseteq S$$

$$x_j \in \mathbb{Z}_+, \qquad \text{para } p_j \in \Pi$$

Apesar dessa formulação ignorar o desperdício, o mesmo pode ser controlado restringindo o conjunto S aos padrões de corte com pequena perda por exemplo.

O processo de busca local consiste em, partindo de uma solução inicial, encontrar uma solução melhor na vizinhança dessa solução inicial, obtendo assim um ótimo local. A busca local iterada (ILS) repete este processo partindo de outras soluções iniciais obtidas por pertubações aleatórias. Define-se como vizinhança neste método, o conjunto de soluções obtidas através da substituição de um padrão por outro diferente. Para calcular o valor da função objetivo na vizinhança é necessário calcular as frequências dos padrões de corte. Para isso, é usado uma heurística baseada no Método de Gauss-Seidel não-linear. Como não é possível trabalhar com todos os padrões de corte viáveis, usa-se uma heurística de geração de padrões de corte (APG).

Capítulo 5

Modelo Não-Linear para o PCOPS

Apresentamos neste capítulo um novo método para resolução do Problema de Corte para Minimizar o número de Objetos Processados e o *Setup* (PCOPS). Ao contrário dos outros métodos que visam minimizar o *setup* e o número de objetos, buscamos desenvolver um método que resolvesse de fato o PCOPS, partindo de sua formulação, utilizando, assim, os custos c_1 (custo unitário de cada objeto) e c_2 (custo do *setup*) na função objetivo a ser minimizada:

$$\text{Minimizar} \quad c_1 \sum_{j=1}^{n} x_i + c_2 \sum_{j=1}^{n} \delta(x_j)$$

$$\text{sujeito a:} \quad \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \qquad i = 1, ..., m.$$

$$x_j \in \mathbb{N}, \qquad j = 1, ..., n.$$

Visto que a função objetivo deste problema de programação inteira é descontínua, o primeiro passo foi suavizá-la. Uma vez "contornado" este problema, como explicaremos

nas seções 5.1 e 5.2, nos deparamos com o desafio de gerar colunas para um problema não-linear. Apresentaremos na seção 5.3 uma proposta que utiliza um problema de programação linear auxiliar. É importante notar que na formulação acima temos uma restrição de desigualdade ao invés de igualdade em relação à demanda. Adotamos esta restrição, pois trabalhar com a desigualdade ampliou a possibilidade de obter soluções com baixo *setup* sem aumentar demasiadamente o número de objetos processados. Com efeito, a restrição de igualdade restringe o espaço das possíveis soluções. Assim, a partir da seção 5.2, quando nos referirmos ao PCOPS está implícito que trabalhamos com restrições de desigualdade em relação à demanda. Isso também não é estranho às aplicações indústriais como vimos no segundo capítulo. No capítulo 6 mostraremos os testes computacionais que indicam que esta combinação da suavização, com um método de programação não-linear, com geração de colunas adaptada a esta formulação e com uma simples heurística de arredondamento, resultaram num novo e competitivo método.

5.1 Minimização de Funções Custo Descontínuas por Suavização

Muitos problemas práticos requerem a minimização de funções que envolvem custos descontínuos. Além do Problema de Corte para Minimizar o número de Objetos Processados e o *Setup* (PCOPS), visto nos capítulos anteriores, podemos citar modelos para expansão de capacidade de redes de telecomunicações [47], problemas de alocação de capacidades e fluxo [29], entre outros. Entre as ferramentas usadas para resolver esses problemas estão o Método Monte Carlo [55], métodos de decomposição [47], técnicas de programação inteira [29], estratégias do gradiente projetado [9] e simulated annealing [35].

Martinez [39] propôs um método que usa uma suavização da função custo descontínua para poder aplicar um método de programação não-linear. Considere o seguinte problema de otimização:

$$\text{Min} \quad f(x) + \sum_{i=1}^{m} H_i[g_i(x)] \quad \text{s.a.} \quad x \in \Omega \tag{5.1}$$

onde $f : \mathbb{R}^n \to \mathbb{R}$ é contínua, $g_i : \mathbb{R}^n \to \mathbb{R}$ é contínua para todo $i = 1, ..., m$, $\Omega \subseteq \mathbb{R}^n$ e $H_i : \mathbb{R} \to \mathbb{R}$, $i = 1, ..., m$, é não-decrescente tal que H_i é contínua exceto nos pontos α_{ij} , $j \in I_i$. O conjunto I_i pode ser finito ou infinito, mas o conjunto dos pontos onde H_i é descontínua é um conjunto discreto, ou seja:

$$\inf\{|\alpha_{il} - \alpha_{ij}| \text{ tal que } l, j \in I_i, l \neq j\} > 0 \tag{5.2}$$

Consideramos também que os limites laterais $\lim_{t \to \alpha_{ij}^-} H_i(t)$, $\lim_{t \to \alpha_{ij}^+} H_i(t)$, existem para todo $j \in I_i$ e:

$$\lim_{t \to \alpha_{ij}^-} H_i(t) = H_i(\alpha_{ij}) < \lim_{t \to \alpha_{ij}^+} H_i(t)$$

para todo $j \in I_i$, $i = 1, ..., m$.

A função custo H_i será aproximada por uma família de funções contínuas não-decrescentes $H_{ik} : \mathbb{R} \to \mathbb{R}$. Assumimos que estas funções aproximativas são tais que, para todo $\mu > 0$,

$$\lim_{k \to \infty} H_{ik}(t) = H_i(t) \text{ uniformemente em } \mathbb{R} \setminus \bigcup_{j \in I_i} (\alpha_{ij}, \alpha_{ij} + \mu). \tag{5.3}$$

Note que (5.3) implica que:

$$\lim_{k\to\infty} H_{ik}(t) = H_i(t) \quad \forall\, t \in \mathbb{R}$$

Para cada k definimos o problema aproximado P_k como:

$$\text{Minimizar} \quad f(x) + \sum_{i=1}^{m} H_{ik}[g_i(x)], \quad \text{sujeito a:} \quad x \in \Omega \tag{5.4}$$

Como (5.4) tem função objetivo contínua, podemos usar algoritmos de otimização contínua para obter sua solução . O teorema seguinte, demonstrado por Martinez em [39], prova que a solução de (5.1) pode ser aproximada pela solução de (5.4).

Teorema 5.1 (Martinez). *Assuma que para todo $k = 0, 1, 2, ...$, x^k é solução de (5.4) e que $x^* \in \Omega$ é um ponto de acumulação de $\{x^k\}$. Então x^* é solução de (5.1).*

Demonstração: Tomando uma sequência apropriada e renomeando-a, podemos assumir, sem perda de generalidade, que $x^k \to x^*$. Como todas as funções g_i são contínuas, isto implica que:

$$g_i(x^k) \to g_i(x^*), \quad \forall \;\; i = 1, ..., m. \tag{5.5}$$

Suponha que $i \in \{1, ..., m\}$ é tal que:

$$g_i(x^*) \notin \{\alpha_{ij},\; j \in I_i\}.$$

Defina

$$\mu = \frac{1}{2}\ \min\{|g_i(x^*) - \alpha_{ij}|,\ j \in I_i\}.$$

A hipótese (5.2) garante que $\mu > 0$. Dado $\varepsilon > 0$, por (5.3), existe k_0 tal que, para todo $k \geq k_0$,

$$|H_{ik}(t) - H_i(t)| \leq \varepsilon/2,\ \ \forall\ t \in \mathbb{R} \setminus \bigcup_{j \in I_i} (\alpha_{ij}, \alpha_{ij} + \mu). \tag{5.6}$$

Pela continuidade de g_i, existe k_1 tal que:

$$g_i(x^k) \in \mathbb{R} \setminus \bigcup_{j \in I_i} (\alpha_{ij}, \alpha_{ij} + \mu). \tag{5.7}$$

Então, por (5.6) temos:

$$\left|H_{ik}[(g_i(x^k)] - H_i[(g_i(x^k)]\right| \leq \varepsilon/2,\ \ \forall\ k \geq\ \max\{k_0, k_1\}. \tag{5.8}$$

Como $g_i(x^k) \to g_i(x^*)$ e $g_i(x^*) \in \{\alpha_{ij}\ ,\ j \in I_i\}$, temos que H_i é contínua em $g_i(x^*)$. Portanto, existe k_2 tal que:

$$\left|H_{ik}[(g_i(x^k)] - H_i[(g_i(x^k)]\right| \leq \varepsilon/2,\ \ \forall\ k \geq k_2. \tag{5.9}$$

Por (5.8) e (5.9) temos que, para $k \geq \max\{k_0, k_1, k_2\}$:

$$\left|H_{ik}[(g_i(x^k)] - H_i[(g_i(x^k)]\right| \leq \left|H_{ik}[(g_i(x^k)] - H_i[(g_i(x^k)]\right| + \left|H_i[(g_i(x^k)] - H_i[(g_i(x^*)]\right| \leq \varepsilon. \tag{5.10}$$

Provamos assim que, se $g_i(x^*) \notin \{\alpha_{ij},\ j \in I_i\}$:

$$\lim_{k\to\infty} H_{ik}[g_i(x^k)] = H_i[g_i(x^*)]. \tag{5.11}$$

Suponha agora que $g_i(x^*) = \alpha_j$ para algum $j \in I_i$. A sequência $\{H_{ik}[g_i(x^k)]\}$ pode ser decomposta em duas subsequências: uma que corresponde aos índices k tal que $g_i(x^k) \leq \alpha_{ij}$ (subsequência esquerda) e outra correspondente aos índices k tal que $g_i(x^k) > \alpha_{ij}$ (subsequência direita). Uma delas pode ser finita. Suponhamos que a subsequência esquerda é infinita. Dado um $\varepsilon > 0$ arbitrário, tome k_0 como antes. Como na prévia escolha de k_1, existe k_3 tal que para todos os índices k da subsequência esquerda com $k \geq k_3$ vale (5.7). Portanto, para estes índices $k \geq k_3$ temos:

$$\left|H_{ik}[(g_i(x^k)] - H_i[(g_i(x^k)]\right| \leq \varepsilon/2$$

Pela continuidade a esquerda de H_i em α_{ij}, existe k_4 tal que para todos os índices k da subsequência esquerda tal que $k \geq k_4$, temos:

$$\left|H_i[(g_i(x^k)] - H_i[(g_i(x^*)]\right| \leq \varepsilon/2$$

Combinando as duas desigualdades anteriores, podemos concluir que, para todos os índices da subsequência esquerda tal que $k \geq \max\{k_0, k_3, k_4\}$,

$$\left|H_{ik}[(g_i(x^k)] - H_i[(g_i(x^*)]\right| \leq \varepsilon$$

Logo, para subsequência esquerda:

$$\lim_{k\to\infty} H_{ik}[g_i(x^k)] = H_i[g_i(x^*)]. \tag{5.12}$$

Para a subsequência direita, como $g_i(x^k) \geq g_i(x^*)$, temos que

$$H_{ik}[g_i(x^k)] \geq H_{ik}[g_i(x^*)]$$

visto que H_{ik} é não-decrescente. Portanto,

$$\liminf_{k\to\infty} H_{ik}[g_i(x^k)] \geq \liminf_{k\to\infty} H_{ik}[g_i(x^*)] = \lim_{k\to\infty} H_{ik}[g_i(x^*)] = H_i[g_i(x^*)]. \tag{5.13}$$

Por (5.12) e (5.13), temos que, quando $g_i(x^*) = \alpha_{ij}$ para algum $j \in I_i$:

$$\liminf_{k\to\infty} H_{ik}[g_i(x^k)] \geq H_i[g_i(x^*)]. \tag{5.14}$$

Por (5.11) e (5.14) e pela continuidade de f, temos que:

$$\liminf_{k\to\infty} f(x^k) + \sum_{i=1}^{m} H_{ik}[g_i(x^k)] \geq f(x^*) + \sum_{i=1}^{m} H_i[g_i(x^*)] \tag{5.15}$$

Suponhamos agora, por absurdo, que x^* não é solução do problema (5.1). Neste caso, existe $\bar{x} \in \Omega$ e $c > 0$ tal que:

$$f(\bar{x}) + \sum_{i=1}^{m} H_i[g_i(\bar{x})] < f(x^*) + \sum_{i=1}^{m} H_i[g_i(x^*)] - c \tag{5.16}$$

Mas, para todo $i = 1, ..., m$,

$$\lim_{k\to\infty} H_{ik}[g_i)\bar{x})] = H_i[g_i(\bar{x})],$$

logo,

$$\lim_{k\to\infty} f(\bar{x}) + \sum_{i=1}^{m} H_{ik}[g_i)\bar{x})] = f(\bar{x}) + \sum_{i=1}^{m} H_i[g_i(\bar{x})].$$

Portanto, por (5.16), existe k_5 tal que:

$$f(\bar{x}) + \sum_{i=1}^{m} H_{ik}[g_i)\bar{x})] < f(x^*) + \sum_{i=1}^{m} H_i[g_i(x^*)] - c/2$$

para todo $k \geq k_5$. Assim, por (5.15), para $k > k_5$ temos que:

$$f(\bar{x}) + \sum_{i=1}^{m} H_{ik}[g_i(\bar{x})] < \liminf_{k\to\infty} f(x^k) + \sum_{i=1}^{m} H_i[g_i(x^k)] - c/2$$

Como x^k é solução de (5.4), isto implica que:

$$f(\bar{x}) + \sum_{i=1}^{m} H_{ik}[g_i(\bar{x})] < \liminf_{k\to\infty} f(\bar{x}) + \sum_{i=1}^{m} H_i[g_i(\bar{x})] - c/2$$

para todo $k \geq k_5$. Isto contradiz a hipótese de que x^* não é solução de (5.1). Como queríamos demonstrar.

5.2 Suavizando o PCOPS

Como já observamos, a partir de agora trabalharemos com o Problema de Corte para Minimizar o número de Objetos Processados e o *Setup* (PCOPS) com restrições de desigualdade em relação a demanda, ou seja, a demanda precisa ser atendida mas não exatamente. O custo de um eventual, na verdade provável, excesso de produção está automaticamente incluso na função objetivo que visa minimizar o número de objetos e o *setup*. Porém, é importante salientar que os resultados da seção anterior, onde apresentamos os problemas com restrições de igualdade, continuam valendo, visto que basta acrescentar variáveis de folga para obter uma formulação com restrições de igualdade.

O primeiro passo para suavizar o PCOPS é relaxar as condições de integralidade. Assim, utilizando a notação da seção anterior, trabalharemos com o seguinte problema de minimização que chamamos de PCOPSR:

$$\text{Minimizar} \quad c_1 f(x) + c_2 \sum_{i=1}^{n} H_i(x)$$

$$\text{sujeito a:} \quad x \in \Omega$$

onde:

- $\Omega = \{x \in \mathbb{R}^n \text{ tal que } Ax \geq d,\ x \geq 0\}$;
- $f(x) = \sum_{i=1}^{n} x_i$;
- $H_i(t) = \delta(t),\ i = 1, ..., n.$

Temos $I_i = \{0\}$ para todo $i = 1, ..., n$, visto que $t = 0$ é o único ponto de descontinuidade de $H_i(t)$, $i = 1, ..., n$. Vale observar que:

$$\lim_{t\to 0^-} H_i(t) = 0 = H_i(0) < \lim_{t\to 0^+} H_i(t) = 1$$

Aproximaremos cada uma das funções H_i pelas seguintes funções contínuas:

$$H_{ik}(t) = \begin{cases} 0 & \text{se } t \leq 0 \\ kt^2/(1+kt^2) & \text{se } t > 0 \end{cases}$$

É fácil ver que $\lim_{k\to\infty} H_{ik}(t) = H_i(t)$ para todo $t \in \mathbb{R}$, para todo $i = 1, ..., n$ e que $H_{ik}(t)$ converge uniformemente para $H_i(t)$ se $t \neq 0$. As figuras 5.1 e 5.2 ilustram esta propriedade.

Assim, estamos nas condições do teorema (5.1). Seja P_k o seguinte problema de programação não-linear que chamaremos de Problema Aproximado:

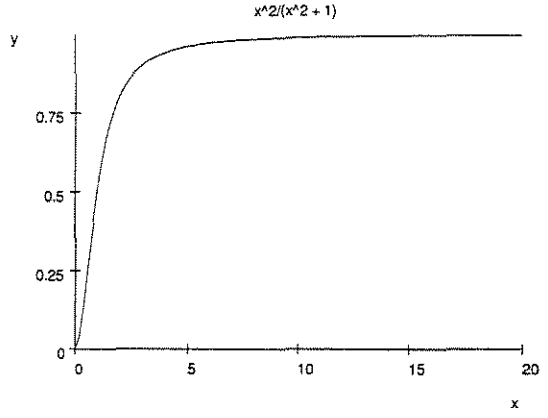
$$\text{Minimizar} \quad c_1 \sum_{j=1}^{n} x_i + c_2 \sum_{i=1}^{n} \frac{kx_j^2}{1+kx_j^2}$$

$$\text{sujeito a:} \quad \sum_{j=1}^{n} a_{ij}x_j \geq d_i, \qquad i = 1, ..., m.$$

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

Logo, pelo teorema (5.1), se para todo $k = 0, 1, 2, ...$, x_k é solução do Problema Aproximado P_k e x^* é ponto de acumulação da sequência $\{x_k\}$, então x^* é solução do PCOPSR.

Uma vez obtida a solução do PCOPSR, podemos usar uma heurística de arredondamento para obter uma solução do PCOPS. Contudo, existe um problema prévio a ser



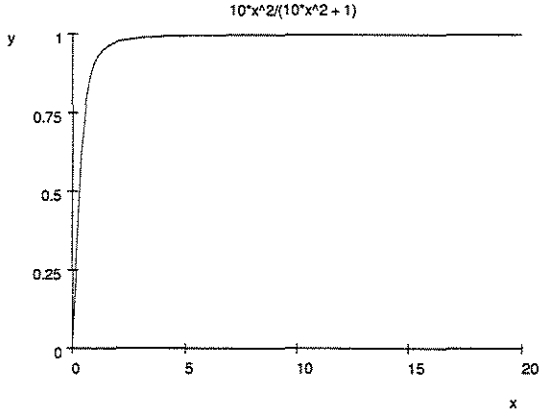


Figura 5.1: Gráficos da função H_{ik} (k=1 e k=10)

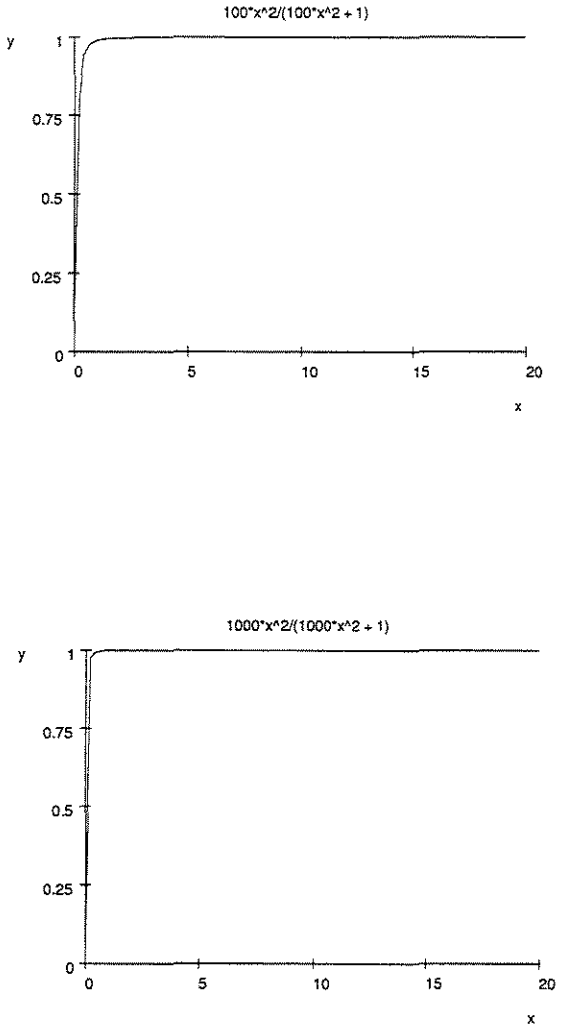


Figura 5.2: Gráficos da função H_{ik} (k=100 e k=1000)

resolvido: que colunas usar no PCOPSR? Apresentamos uma proposta na próxima seção.

5.3 Gerando Colunas em um Problema Não-Linear

Como observamos nos primeiros capítulos, não é possível trabalhar com todos os padrões de corte viáveis. Com isso, buscamos adaptar o Método de Geração de Colunas para o problema aproximado P_k, que é um problema de programação não-linear que aproxima a solução do PCOPS. Assim, a idéia foi encontrar um problema de programação linear que possibilitasse o uso do método de geração de colunas.

Considere o seguinte problema de programação linear auxiliar que chamaremos de PLA1 :

$$\text{Min} \quad \sum_{i=1}^{n} x_i - \sum_{i=1}^{P} y_i \tag{5.17}$$

$$\text{sujeito a:} \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \quad i = 1, 2, ..., m \tag{5.18}$$

$$\sum_{j=1}^{n} y_j = P \tag{5.19}$$

$$L y_j - x_j \geq 0, \quad j = 1, 2, ..., n \tag{5.20}$$

$$0 \leq y_j \leq 1, \quad j = 1, 2, ..., n \tag{5.21}$$

$$x_j \geq 0, \quad j = 1, 2, ..., n \tag{5.22}$$

onde:

- P é o número de padrões distintos presentes na solução oriunda de P_k;

- L é um número real grande.

A equação (5.20) determina que, se $y_j = 0$ então $x_j = 0$. Como a função objetivo (5.17) tende a tornar $y_j = 1$ para $j = 1, ..., P$, a restrição (5.19) garante, neste caso, que $(n-P)$ componentes de y se anulam, mais especificamente $y_{P+1} = y_{P+2} = ... = y_n = 0$, donde concluímos que $x_j = 0$ para $j = P+1, ..., n$. Seja x^* uma solução ótima deste problema. Temos $x^*_{P+1} = ... = x^*_n = 0$. Ou seja, *setup* $\leq P$, onde, lembremos, *setup* é o número de padrões distintos na solução final. Se $\sum_{j=1}^{P} x^*{}_j < \sum_{j=1}^{P} x_j$ nossa hipótese de otimalidade da solução x do PCOPS estaria violada. Logo, $x = x^*$.

Entretanto, é fácil notar que podemos simplificar este procedimento resolvendo o Problema de Programação Linear com apenas as primeiras P variáveis. Esta foi a nossa opção. Assim, utilizamos o seguinte Problema de Programação Linear Auxiliar (PLA2) para gerar colunas:

$$\text{Mininimizar} \quad \sum_{j=1}^{P} x_j$$

$$\text{sujeito a:} \quad \sum_{j=1}^{P} a_{ij} x_j \geq d_i, \quad i = 1, ..., m.$$

$$x_j \geq 0, \qquad j = 1, ..., P$$

No processo de geração de colunas para o PLA2 temos que resolver um Problema da Mochila como observamos no capítulo 3. Haessler sugeriu em [24] que fosse resolvido um Problema da Mochila Limitada (PML) para gerar boas colunas que, além de diminuírem o número de objetos processados, também ajudassem a minimizar o *setup*. Neste artigo, Haessler sugeriu fixar os limitantes da seguinte forma:

$$b_i = \min\left\{\frac{d_i}{10}, \left\lfloor\frac{W}{w_i}\right\rfloor\right\}, \quad i = 1, ..., m$$

Incorporamos esta estratégia de gerar colunas através da resolução de um PML no nosso método, utilizando porém, outra heurística, por nós desenvolvida, para gerar estes limitantes b_i, $i = 1, ..., m$, conforme explicaremos no próximo capítulo. Isto porque consideramos "demasiadamente heurístico" dividir por 10 a demanda para obter os limitantes para cada item.

5.4 O Novo Método

Antes de apresentar o novo método, vale recordar a formulação do PCOPS com restrições de desigualdade:

$$\text{Minimizar} \quad c_1 \sum_{j=1}^{n} x_i + c_2 \sum_{j=1}^{n} \delta(x_j)$$

$$\text{sujeito a:} \quad \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \qquad i = 1, ..., m.$$

$$x_j \in \mathbb{N}, \qquad j = 1, ..., n.$$

Podemos agora descrever o novo método (NM) mais detalhadamente:

Novo Método

1. Gere uma solução inicial heuristicamente para o PCOPS;

2. Faça a melhor solução encontrada (x^*) igual a solução inicial;

3. Obtenha uma solução para o PCOPS corrente resolvendo P_k, com restrições de desigualdade, para $k = 10^{i-1}$, $i = 1, 2, ..., 5$;

4. Se a solução encontrada em (3) é melhor que (x^*) atualize-a;

5. Obtenha os multiplicadores simplex π_i, $i = 1, ..., m$, resolvendo o PLA2;

6. Resolva o problema da mochila limitada (PML) formada pelos multiplicadores simplex de PLA2:

$$\text{Maximizar} \quad Z = \sum_{i=1}^{m} \pi_i x_i$$

$$\text{sujeito a:} \quad \sum_{i=1}^{m} w_i x_i \leq W$$

$$x_i \leq b_i, \qquad i = 1, ..., m$$

$$x_i \in \mathbb{N} \qquad i = 1, ..., m.$$

7. Se $Z \leq 1$ resolva o Problema da Mochila sem limitantes;

8. Se $Z \leq 1$ $\rightarrow$FIM. Caso contrário insira a coluna gerada na matriz do PCOPS e volte a (3).

9. Use um método de arredondamento para obter uma solução inteira.

Vale observar que o procedimento de geração de colunas que propomos é heurístico, ou seja, não podemos garantir que tenha o desempenho que desejamos em todos os casos. Até porque não necessariamente encontramos a solução ótima global do P_k. O mais provável é obtermos ótimos locais. Porém, os testes computacionais mostraram que nossa proposta obtém bons resultados.

No próximo capítulo detalharemos como implementamos cada passo do novo método. Descreveremos também os resultados computacionais, nossas conclusões e perspectivas.

Capítulo 6

Implementação e Resultados Computacionais

6.1 Implementação do Método NM

Para obter a solução inicial que utilizamos no método NM, implementamos o método SHP descrito no capítulo 4 com algumas pequenas modificações:

- ao invés de fixar o parâmetro $MAXTL$, que é o limitante de desperdício no padrão que está sendo gerado, inicializamos com um valor baixo, $MAXTL = 0,01W$, pois no início do processo é possível gerar padrões com despedício muito pequeno, e a cada padrão gerado aumentamos o parâmetro em $0,25\%$, ou seja, fazemos $MAXTL = MAXTL + 0,0025W$, até atingir o limite de 3% de W.
- Também não fixamos o valor de $MINU$. Inicializamos com $MINU = 0,5NR$ e a cada novo padrão gerado somamos $0,05NR$ até o máximo de $0,9NR$. Isso porque desejamos gerar uma solução com baixo desperdício e um *setup* não muito alto, pois o nosso método certamente melhora o *setup*. Como quanto mais alto o $MINU$

provavelmente menor será o *setup* e maior o desperdício, resolvemos ampliar o espaço de busca nas primeiras iterações fixando o parâmetro $MINU$ no menor valor sugerido por Haessler ($MINU = 0,5NR$), e posteriormente, aumentá-lo 5% por iteração, para não comprometer demasiadamente o *setup*.

- No passo 5 do algoritmo SHP é feita a busca em ordem lexicográfica com intuito de obter um padrão que satisfaça os critérios de aspiração. É válido explicar nossa implementação deste passo. Iniciamos alocando no padrão que está sendo gerado o maior número possível do item com maior demanda residual. Depois tentamos, seguindo a ordem decrescente em relação à demanda residual, alocar o maior número possível de itens diferentes. No caso do padrão gerado não satisfazer os critérios de aspiração diminuímos em uma unidade a frequência do item com maior demanda residual e repetimos a busca. Adotamos esta estratégia heurística por acreditar que, alocando o maior número de itens diferentes no padrão, poderíamos contribuir para uma solução final com baixo *setup*. É o inverso do que ocorre na heurística FFD, onde tenta-se alocar o maior número possível de cada um dos itens no padrão que é gerado. Como observamos no capítulo 3, a heurística FFD gera uma boa solução inicial, com baixo desperdício, mas com $Setup = m$. Com a heurística SHP buscamos um *setup* menor, e por isso optamos por uma busca que "privilegiasse" padrões com baixo *setup*, já que o baixo desperdício já é "garantido" pelo parâmetro $MAXTL$.

Já o parâmetro $MINR$ é fixado conforme sugestão de Haessler em $MINR = \overline{NI} - 1$. O parâmetro $MAXR$, número máximo de facas, não foi utilizado.

Utilizamos o BOX9903 [32, 58], desenvolvido pelo Grupo de Otimização do Departamento de Matemática Aplicada da Unicamp, para resolver a sequência de problemas P_k (o BOX9903 está disponível na página do professor Martinez [58]). Esta sub-rotina resolve o seguinte problema de otimização:

$$\text{Minimizar} \quad f(x)$$

$$\text{sujeito a:} \quad Ax = d$$

$$h(x) = 0$$

$$l \leq x \leq u$$

onde $f : \mathbb{R}^n \rightarrow \mathbb{R}$ e $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$ são diferenciáveis e A é uma matriz $m \times n$. No nosso caso não temos a função $h(x)$, $l = 0$ e $u = \infty$. Como nosso problema P_k tem restrições de desigualdade, precisamos acrescentar variáveis de folga para utilizarmos o BOX9903 que foi desenvolvido para trabalhar com restrições de igualdade. Obtemos então uma nova matriz $\overline{A}$. Assim, em cada iteração o BOX9903 resolve o seguinte problema:

$$\text{Minimizar} \quad L(x)$$

$$\text{sujeito a:} \quad l \leq x \leq u$$

onde:

$$L(x) = c_1 \sum_{i=1}^{n} x_i + c_2 \sum_{i=1}^{n} h_{ik}(x_i) + \lambda^t \cdot (\overline{A}x - d) + (\rho/2) \cdot \left\|\overline{A}x - d\right\|_2^2$$

é chamado de Lagrangiano Aumentado. Na iteração j temos $\rho = 10^j$ e $k = 10^{j-1}$, sendo que j assume os valores $1, 2, 3, 4$ e 5, ou seja, utilizamos 5 iterações do método. É importante ressaltar que o parâmetro k do problema aproximado aumenta a cada iteração do método do lagrangiano aumentado. Nos testes verificamos que isso em

nada alterava a qualidade da solução e trouxe ainda um grande ganho de tempo computacional. O vetor de multiplicadores de lagrange λ é estimado em cada iteração j através da seguinte expressão:

$$\lambda_j = \lambda_{j-1} + \rho_j(\overline{A}x_j - b)$$

Vale lembrar que:

$$h_{ik}(x) = kx_i^2/(1 + kx_i^2)$$

Como o método não necessariamente encontra o ótimo global, a cada nova coluna adicionada à matriz de restrições A, resolvemos a sequência de Problemas P_k, $k = 10^{i-1}$ para $i = 1, 2, 3, 4, 5$, com 20 soluções iniciais diferentes: a solução corrente, a solução nula e outras 18 geradas aleatoriamente. A solução corrente do P_k é a melhor solução encontrada. Para fazer a geração aleatória dessas 18 soluções iniciais utilizamos uma proposta de Martinez presente no código *startcos*, disponíbilizado em sua página [58]. No apêndice C transcrevemos o pseudo-código deste gerador de soluções iniciais.

Após resolver a sequência de problemas P_k, devemos gerar uma coluna para entrar na matriz de restrições. Tentamos inicialmente gerar uma boa coluna também para o *setup*, resolvendo assim um Problema da Mochila Limitada conforme descrevemos no capítulo anterior. Para isso é necessário definir os limitantes. Objetivando diminuir o *setup* em 20% desenvolvemos uma nova heurística para obter estes limitantes. Sendo o $Setup = P$, ou seja, existindo P padrões distintos na solução corrente, o nosso objetivo é obter, após a geração de colunas, $Setup = 0.8P$. Seja NB o número de objetos processados na solução corrente. Supondo que este fique constante, devemos ter em média $MBP = NB/Setup$ objetos processados por padrão. Suponhamos que o novo padrão a ser gerado entre na solução com este valor, isto é, $x_{n+1} = MBP$, e que os

itens alocados neste padrão não estejam presentes em outros padrões não-nulos, o que seria razoável, já que desejamos diminuir o *setup*. Neste caso, para não excedermos a demanda, cada item i deveria ser limitado por $b_i = d_i/MBP$. Como precisamos de um limitante inteiro, arredondamos b_i para baixo. E caso $b_i > W/w_i$ fazemos $b_i = \lfloor W/w_i \rfloor$, que é o limitante relativo à largura da bobina, uma vez que admitimos apenas padrões viáveis.

Trabalhamos, desta forma, com estes limitantes para gerar uma nova coluna através do método de Gilmore e Gomory resolvendo um Problema da Mochila Limitada. Para isso, utilizamos o algoritmo de Martello e Toth MTB2 e para resolver o Problema da Mochila Ilimitada (PMI) utilizamos o algoritmo *branch-and-bound*, conforme descrevemos no capítulo 3.

Por fim, para arredondar a solução encontrada ao final do processo, utilizamos o método BRURED, descrito no capítulo 3, pois este não altera o número de padrões distintos e é extremamente rápido do ponto de vista computacional.

Podemos assim, detalhar a implementação do método NM:

Implementação do Método NM

1. Gere uma solução inicial através da heurística SHP ;
2. Faça a melhor solução encontrada (x^*) igual a solução inicial;
3. Utilizando o BOX9903 obtenha uma solução para o PCOPS corrente resolvendo P_k, com restrições de desigualdade, para $k = 10^{i-1}$, $i = 1, 2, ..., 5$, com 20 soluções iniciais diferentes;
4. Se a solução encontrada em (3) é melhor que x^* atualize-a;
5. Obtenha os multiplicadores simplex π_i, $i = 1, ..., m$, resolvendo-se PLA2;

6. Resolva o problema da mochila limitada (PML), através o método de Martello e Toth MTB2, obtendo o valor da função objetivo Z e a coluna a solução do PML;

7. Se $Z \leq 1$ resolva o Problema da Mochila Ilimitada através do método Branch-and-Bound;

8. Se $Z \leq 1$ →FIM. Caso contrário insira a coluna gerada na matriz do PCOPS e volte a (3).

9. Use o método BRURED para obter uma solução inteira.

6.2 Testes Computacionais

Geramos, através do gerador CUTGEN1 desenvolvido por Gau e Wascher [20], 100 problemas para cada uma das 18 classes caracterizadas pelos diferentes valores para o limite inferior e superior do comprimento dos itens (v_1 e v_2 do CUTGEN1), demanda média (DBar de CUTGEN1) e número de itens demandados (m de CUTGEN1). Mais especificamente:

- v_1 assumiu os valores $0,01$ e $0,2$;
- v_2 assumiu os valores $0,2$ e $0,8$;
- O número de itens requeridos foi fixado em 10, 20 ou 40;
- A demanda média dos itens foi 10 ou 100.

Obtemos desta forma, seis classes com itens pequenos ($v_1 = 0,01$ e $v_2 = 0,2$), seis com itens de larguras variadas ($v_1 = 0,01$ e $v_2 = 0,8$) e outras seis classes com itens grandes ($v_1 = 0,2$ e $v_2 = 0,8$), conforme a tabela 6.1.

Classe	v_1	v_2	m	d
1	0,01	0,2	10	10
2	0,01	0,2	10	100
3	0,01	0,2	20	10
4	0,01	0,2	20	100
5	0,01	0,2	40	10
6	0,01	0,2	40	100
7	0,01	0,8	10	10
8	0,01	0,8	10	100
9	0,01	0,8	20	10
10	0,01	0,8	20	100
11	0,01	0,8	40	10
12	0,01	0,8	40	100
13	0,2	0,8	10	10
14	0,2	0,8	10	100
15	0,2	0,8	20	10
16	0,2	0,8	20	100
17	0,2	0,8	40	10
18	0,2	0,8	40	100

Tabela 6.1: Classes Geradas

Comparamos os nossos resultados com os obtidos usando a nossa implementação da heurística SHP e os obtidos pela heurística Kombi234, desenvolvida por Foester e Wascher [18], conforme descrevemos no capítulo 4. Estes métodos também foram usados por Umetami et al. [49] para efeitos de comparação da sua heurística ILS-APG. Adotamos no gerador CUTGEN1 a mesma semente, 1994, usadas por Foester e Wascher e Umetami para gerar os 1800 problemas testes.

Estamos comparando o método NM com o heurística Kombi234 cuja solução inicial foi obtida através do procedimento de Stadler [46] que descrevemos no capítulo 3: utiliza-se o método de geração de colunas e depois usa-se o método CSTAOPT para obter uma solução inteira viável. Isto porque foi esta "combinação", *Procedimento de Stadler + Kombi234*, que obteve os melhores resultados até então na literatura, segundo os próprios autores Foester e Wascher, no mínimo até a publicação desse trabalho [18].

Implementamos o método NM e a heurística SHP na linguagem Fortran, com compilador g77 para Linux, num micro-computador com processador Athlon XP 1800Mhz, 512MB de RAM. Os resultados da heurística Kombi234 foram obtidos num código em MODULA-2 em MS-DOS 6.0 usando um IBM 486/66, implementado pelos próprios autores Foester e Wascher [18].

Apresentamos os resultados do método NM com dois valores para o custo do *setup*: $c_2 = 100$, que chamaremos de NM100 e $c_2 = 300$, que chamaremos de NM300. Nos dois casos fixamos o custo de cada objeto processado em uma unidade, ou seja, $c_1 = 1$. Estes valores de c_2 cumprem o papel de penalizar o número de padrões distintos (*setup*). Desta forma, NM300 obtém um *setup* menor que o NM100 na maioria das classes. Visto que *setup* e número de objetos processados são objetivos conflitantes, o NM100 apresentou número de objetos menor que o NM300 na maioria das classes.

6.2.1 Comparando o *setup* e o número de objetos processados

As tabelas 6.2, 6.3 e 6.4 apresentam as médias do número de objetos e do *setup* para 100 problemas de cada uma das classes, divididas, respectivamente, em classes com itens pequenos, com itens de largura variadas e com itens grandes.

A tabela 6.5 apresenta a variação do *setup* e do número de objetos do método NM100 em relação à heurística SHP. Para o cálculo da variação do *setup* utilizamos

	SHP		Kombi		NM100		NM300	
Classe	*setup*	Objetos	*setup*	Objetos	*setup*	Objetos	*setup*	Objetos
1	3,95	14,17	3,40	11,49	3,14	18,44	3,02	14,30
2	5,94	116,47	7,81	110,25	4,66	116,66	4,50	121,48
3	5,00	25,29	5,89	22,13	4,88	25,18	4,84	25,14
4	7,31	225,33	14,26	215,93	7,16	226,72	7,28	224,86
5	6,87	46,89	10,75	42,96	7,02	45,64	7,02	45,66
6	10,81	433,59	25,44	424,71	10,96	432,68	10,92	432,72
Média	6,65	143,62	11,26	137,91	6,30	144,22	6,26	144,03

Tabela 6.2: Médias para classes com itens pequenos

	SHP		Kombi		NM100		NM300	
Classe	*setup*	Objetos	*setup*	Objetos	*setup*	Objetos	*setup*	Objetos
7	8,84	55,84	7,90	50,21	5,78	50,84	5,54	51,54
8	9,76	515,76	9,96	499,52	8,22	506,02	8,00	495,94
9	17,19	108,54	15,03	93,67	10,90	106,72	10,58	114,52
10	19,37	1001,59	19,28	932,32	14,56	969,40	13,96	969,20
11	32,20	202,80	28,74	176,97	19,80	220,46	20,00	231,84
12	37,25	1873,05	37,31	1766,20	25,58	1813,60	23,68	1861,20
Média	20,77	626,26	19,70	586,48	14,14	611,17	13,63	620,71

Tabela 6.3: Médias para classes com itens de larguras variadas

a expressão $100 \times (Setup_{NM100} - Setup_{SHP})/Setup_{SHP}$. Da mesma forma, para o cálculo da variação do número de objetos utilizamos a expressão: $100 \times (Obj_{NM100} - Obj_{SHP})/Obj_{SHP}$, onde Obj_{NM100} representa a média de objetos processados na solução do NM100.

A média do *setup* do NM100 só não foi melhor do que a obtida pelo SHP nas classes 5 e 6. E a média do número de objetos do NM100 foi melhor que a média do SHP em 12 das 18 classes.

A tabela 6.6 apresenta a variação do *setup* e do número de objetos do método NM100 em relação à heurística Kombi234. Em todas as classes o NM100 obteve uma média de *setup* menor que a heurística Kombi. Já esta obteve média de número de objetos menor que NM100 em todas as classes. Porém, em 7 classes a diferença foi menor ou igual a 3%.

Não apresentamos as mesmas tabelas para os resultados obtidos pelo NM300, pois o desempenho deste é praticamente o mesmo do NM100.

Vale lembrar que o Kombi234, que estamos utilizando para comparação entre os métodos, mantém constante o número de objetos obtidos pelo procedimento de Stadler (CSTAOPT), e este é um dos melhores métodos para minimizar o número de objetos processados. Logo, é natural o Kombi obter média de objetos processados menor que o novo método que propomos.

6.2.2 Comparando o custo total

Nos cálculos da função objetivo para aferição da qualidade do novo método utilizamos valores de c_2 bem abaixo do que o utilizado no algoritmo, onde usamos $c_2 = 100$ no NM100 e $c_2 = 300$ no NM300. Visamos, principalmente, ser justos na comparação

entre os métodos, e também nos aproximar o máximo possível da realidade. Obviamente, quanto mais alto for na prática o valor de c_2 melhor será o "desempenho" do novo método. Mas os resultados mostram que mesmo com valores baixos de c_2, 1, 5 e 10, o novo método obteve custo total médio menor, em geral, que o obtido pelas heurísticas SHP e Kombi. É necessário, contudo, usar valores maiores de c_2 na implementação do método pois estes cumprem o papel de penalizadores do *setup*.

Na literatura, a única menção à esta escolha de valores para c_1 e c_2 que encontramos foi no artigo de Diegel et al. [12]. Porém, esta foi bem generalista. Segundo os autores deste artigo, a relação exata entre os custos de cada objeto processado e o *setup* depende dos "dados em mão", mas sempre são baixos, ou seja, nunca c_2 é muito maior que c_1. Eles chegam a afirmar que $c_1 = c_2 = 1$ é apropriado para problemas grandes. Todavia, quando minimizar o *setup* é o primeiro objetivo, o custo do *setup* deve ser superior ao custo de cada objeto processado. Enfim, como dissemos no segundo capítulo, a relação entre estes dois custos depende de vários fatores como demanda, tempo de entrega, custo da mão de obra, que são custos que variam com o tempo e de indústria para indústria.

Calculando o valor médio do custo total, buscamos analisar o desempenho dos métodos em relação aos dois objetivos simultaneamente, ou seja, visamos comparar o valor da função objetivo do nosso PCOPS obtido pelos diferentes métodos. A tabela 6.7 traz o valor do custo médio obtido por cada um dos métodos quando usamos $c_1 = c_2 = 1$.

Também para uma melhor compreensão do comportamento dos métodos, a tabela 6.8 apresenta a diferença em percentagem do custo médio obtido pelo método NM300 em relação aos método SHP e Kombi quando $c_1 = c_2 = 1$.

Nota-se que o NM300 obteve médias de custo menor que o SHP em 14 classes, e menor que o Kombi em 6, mesmo quando $c_1 = c_2 = 1$. Não apresentamos a mesma

tabela para o NM100, pois os resultados são análogos aos apresentados na tabela 6.8.

Apresentamos na tabela 6.9 o custo total médio obtido por cada método quando $c_1 = 1$ e $c_2 = 5$.

Para uma melhor comparação da média de custo total obtido por cada método, a tabela 6.10 traz a diferença em percentagem do Método NM100 em relação aos métodos SHP e Kombi.

Nota-se que o NM100 obteve uma média de custo menor que o SHP em 16 classes e menor que o Kombi em 14 classes.

Os resultados obtidos pelo NM300, apresentados na tabela 6.11, foram análogos. O NM300 obteve média de custo menor que o SHP em todas as 18 classes e menor que o Kombi em 12 classes.

Apresentamos na tabela 6.12 as médias de custos totais no caso de $c_2 = 10$ e $c_1 = 1$. Para uma melhor comparação entre os métodos repetimos o procedimento adotado nos casos anteriores. A tabela 6.13 apresenta a percentagem de variação do método NM100 em relação aos métodos SHP e Kombi.

Os resultados confirmam o bom desempenho do método NM. Em 16 classes o NM100 apresentou média de custo menor que o SHP e em 17 menor que o Kombi.

Já o NM300, conforme mostra a tabela 6.14, obteve média de custo menor que o SHP em 16 classes e menor que o Kombi em todas as classes. Vale notar ainda que o SHP obteve média de custo apenas $0,23\%$ e $0,04\%$ menor que o NM300 nas classes 5 e 6 respectivamente.

Por fim, apresentamos na tabela 6.15 os tempos computacionais médios de cada método para cada uma das 18 classes.

	SHP		Kombi		NM100		NM300	
Classe	*setup*	Objetos	*setup*	Objetos	*setup*	Objetos	*setup*	Objetos
13	9,38	69,97	8,97	63,27	5,86	66,52	5,64	70,54
14	9,85	643,55	10,32	632,12	7,92	633,98	7,92	634,02
15	18,03	136,03	16,88	119,43	10,28	130,20	9,92	127,38
16	19,63	1253,55	19,91	1191,80	15,00	1193,54	13,88	1194,74
17	34,39	256,01	31,46	224,68	23,32	283,88	22,60	297,58
18	38,23	2381,54	38,28	2342,40	29,80	2410,82	27,44	2430,04
Media	21,59	790,11	20,97	762,28	15,36	786,49	14,57	792,05

Tabela 6.4: Médias para classes com itens grandes

	NM100 em relação ao SHP	
Classe	Variação do *setup*	Variação do número de Objetos
1	-20,51	30,13
2	-21,55	0,16
3	-2,40	-0,43
4	-2,05	0,62
5	2,18	-2,67
6	1,39	-0,21
7	-34,62	-8,95
8	-15,78	-1,89
9	-36,59	-1,68
10	-24,83	-3,21
11	-38,51	8,71
12	-31,33	-3,17
13	-37,53	-4,93
14	-19,59	-1,49
15	-42,98	-4,29
16	-23,59	-4,79
17	-32,19	10,89
18	-22,05	1,23

Tabela 6.5: Variação em % do NM100 em relação ao SHP

	NM100 em relação ao Kombi	
Classe	Variação do *setup*	Variação do número de Objetos
1	-7,65	60,49
2	-40,33	5,81
3	-17,15	13,78
4	-49,79	5,00
5	-34,70	6,24
6	-56,92	1,88
7	-26,84	1,25
8	-17,47	1,30
9	-27,48	13,93
10	-24,48	3,98
11	-31,11	24,57
12	-31,44	2,68
13	-34,67	5,14
14	-23,26	0,29
15	-39,10	9,02
16	-24,66	0,15
17	-25,87	26,35
18	-22,15	2,92

Tabela 6.6: Variação em % do NM100 em relação ao Kombi

Classe	SHP	Kombi	NM100	NM300
1	18,12	14,89	21,58	17,32
2	122,41	118,06	121,32	125,98
3	30,29	28,02	30,06	29,98
4	232,64	230,19	233,88	232,14
5	53,76	53,71	52,66	52,68
6	444,40	450,15	443,64	443,64
7	64,68	58,11	56,62	57,08
8	525,52	509,48	514,24	503,94
9	125,73	108,70	117,62	125,10
10	1020,96	951,60	983,96	983,16
11	235,00	205,71	240,26	251,84
12	1910,30	1803,51	1839,18	1884,88
13	79,35	72,24	72,38	76,18
14	653,40	642,44	641,90	639,94
15	154,06	136,31	140,48	137,30
16	1273,18	1211,71	1208,54	1208,62
17	290,40	256,14	307,20	320,18
18	2419,77	2380,68	2440,62	2457,48

Tabela 6.7: Médias do custo total com $c_1 = c_2 = 1$

Classe	Custo do NM300 em relação	
	SHP	Kombi
1	-4,42	16,32
2	2,92	6,71
3	-1,02	7,00
4	-0,21	0,85
5	-2,01	-1,92
6	-0,17	-1,45
7	-11,75	-1,77
8	-4,11	-1,09
9	-0,50	15,09
10	-3,70	3,32
11	7,17	22,42
12	-1,33	4,51
13	-3,99	5,45
14	-2,06	-0,39
15	-10,88	0,73
16	-5,07	-0,26
17	10,25	25,00
18	1,56	3,23

Tabela 6.8: Variação em % do custo total, com $c_1 = c_2 = 1$, do NM300 em relação ao SHP e Kombi

Classe	SHP	Kombi	NM100	NM300
1	33,92	28,49	34,14	29,40
2	146,17	149,30	139,96	143,98
3	50,29	51,58	49,58	49,34
4	261,88	287,23	262,52	261,26
5	81,24	96,71	80,74	80,76
6	487,64	551,91	487,48	487,32
7	100,04	89,71	79,74	79,24
8	564,56	549,32	547,12	535,94
9	194,49	168,82	161,22	167,42
10	1098,44	1028,72	1042,20	1039,00
11	363,80	320,67	319,46	331,84
12	2059,30	1952,75	1941,50	1979,60
13	116,87	108,12	95,82	98,74
14	692,80	683,72	673,58	671,62
15	226,18	203,83	181,60	176,98
16	1351,70	1291,35	1268,54	1264,14
17	427,96	381,98	400,48	410,58
18	2572,69	2533,80	2559,82	2567,24

Tabela 6.9: Médias do custo total com $c_1 = 1$ e $c_2 = 5$

Classe	Custo do NM100 em relação	
	SHP	Kombi
1	0,65	19,83
2	-4,25	-6,26
3	-1,41	-3,88
4	0,24	-8,60
5	-0,62	-16,51
6	-0,03	-11,67
7	-20,29	-11,11
8	-3,09	-0,40
9	-17,11	-4,50
10	-5,12	1,31
11	-12,19	-0,38
12	-5,72	-0,58
13	-18,01	-11,38
14	-2,77	-1,48
15	-19,71	-10,91
16	-6,15	-1,77
17	-6,42	4,84
18	-0,50	1,03

Tabela 6.10: Variação em % do custo total, com $c_1 = 1$ e $c_2 = 5$, do NM100 em relação ao SHP e Kombi

Classe	Custo do NM300 em relação	
	SHP	Kombi
1	-13,33	3,19
2	-1,50	-3,56
3	-1,89	-4,34
4	-0,24	-9,04
5	-0,59	-16,49
6	-0,07	-11,70
7	-20,79	-11,67
8	-5,07	-2,44
9	-13,92	-0,83
10	-5,41	1,00
11	-8,79	3,48
12	-3,87	1,37
13	-15,51	-8,68
14	-3,06	-1,77
15	-21,75	-13,17
16	-6,48	-2,11
17	-4,06	7,49
18	-0,21	1,32

Tabela 6.11: Variação em % do custo total, com $c_1 = 1$ e $c_2 = 5$, do NM300 em relação ao SHP e Kombi

Classe	SHP	Kombi	NM100	NM300
1	53,67	45,49	49,84	44,50
2	175,87	188,35	163,26	166,48
3	75,29	81,03	73,98	73,54
4	298,43	358,53	298,32	297,66
5	115,59	150,46	115,84	115,86
6	541,69	679,11	542,28	541,92
7	144,24	129,21	108,64	106,94
8	613,36	599,12	588,22	575,94
9	280,44	243,97	215,72	220,32
10	1195,29	1125,12	1115,00	1108,80
11	524,80	464,37	418,46	431,84
12	2245,55	2139,30	2069,40	2098,00
13	163,77	152,97	125,12	126,94
14	742,05	735,32	713,18	711,22
15	316,33	288,23	233,00	226,58
16	1449,85	1390,90	1343,54	1333,54
17	599,91	539,28	517,08	523,58
18	2763,84	2725,20	2708,82	2704,44

Tabela 6.12: Médias do custo total com $c_1 = 1$ e $c_2 = 10$

Classe	Custo do NM100 em relação	
	SHP	Kombi
1	-7,14	9,56
2	-7,17	-13,32
3	-1,74	-8,70
4	-0,04	-16,79
5	0,22	-23,01
6	0,11	-20,15
7	-24,68	-15,92
8	-4,10	-1,82
9	-23,08	-11,58
10	-6,72	-0,90
11	-20,26	-9,89
12	-7,84	-3,27
13	-23,60	-18,21
14	-3,89	-3,01
15	-26,34	-19,16
16	-7,33	-3,40
17	-13,81	-4,12
18	-1,99	-0,60

Tabela 6.13: Variação em % do custo total, com $c_1 = 1$ e $c_2 = 10$, do NM100 em relação ao SHP e Kombi

Classe	Custo do NM300 em relação	
	SHP	Kombi
1	-17,09	-2,18
2	-5,34	-11,61
3	-2,32	-9,24
4	-0,26	-16,98
5	0,23	-23,00
6	0,04	-20,20
7	-25,86	-17,24
8	-6,10	-3,87
9	-21,44	-9,69
10	-7,24	-1,45
11	-17,71	-7,01
12	-6,57	-1,93
13	-22,49	-17,02
14	-4,15	-3,28
15	-28,37	-21,39
16	-8,02	-4,12
17	-12,72	-2,91
18	-2,15	-0,76

Tabela 6.14: Variação em % do custo total, com $c_1 = 1$ e $c_2 = 10$, do NM300 em relação ao SHP e Kombi

Classe	SHP T(s)	Kombi* T(s)	NM500 T(s)	NM300 T(s)
1	0,01	0,14	0,80	1,12
2	0,08	1,14	1,17	3,17
3	0,17	1,74	0,47	0,89
4	0,21	16,00	0,94	1,15
5	0,27	38,03	0,58	0,64
6	0,31	379,17	0,93	0,91
7	0,01	0,07	16,49	18,1
8	0,02	0,2	8,96	11,78
9	0,04	3,37	69,11	105,49
10	0,06	3,25	77,21	106,59
11	0,22	36,26	185,53	216,97
12	0,32	76,31	318,54	376,04
13	0,01	0,08	4,44	7,43
14	0,02	0,13	1,95	2,02
15	0,03	1,81	29,36	60,02
16	0,04	2,6	26,30	39,87
17	0,16	50,93	248,62	267,02
18	0,24	70,94	443,66	538,92

Tabela 6.15: Tempo Médio em segundos de cada Método (*Kombi não foi implementado e testado na mesma máquina e linguagem)

Capítulo 7

Conclusões e Perspectivas

Com base nos resultados computacionais apresentados, podemos afirmar que o método NM é de fato competitivo em relação aos métodos SHP e Kombi. Ao avaliarmos o custo total, $[c_1 \times$ (número de objetos processados) $+ c_2 \times$ (*setup*)], o NM100 e o NM300 chegam a obter médias superiores a 20% melhores do que as obtidas pelas heurísticas SHP e Kombi em várias classes e, como vimos, geralmente melhor em praticamente todas, quando $c_1 = 1$ e $c_2 = 5$ ou 10. O método NM é competitivo mesmo quando $c_1 = c_2 = 1$.

Especificamente em relação ao *setup* o NM100 e NM300 também tem um desempenho melhor que o SHP e o Kombi em praticamente todas as classes.

Vale destacar outra grande vantagem do método NM, que é a possibilidade de se trabalhar com os valores c_1 e c_2 explicitamente na função objetivo. De fato, tais custos são variáveis na prática, e dependem de diversos fatores como demanda, tempo de entrega, custo de mão-de-obra ... Não temos conhecimento de outro método publicado na literatura que trabalhe com os dois custos na função objetivo explicitamente. Mais do que isso, como já afirmamos no início do capítulo 5, não temos conhecimento de

outro método que resolva o problema de minimizar o *setup* e o número de objetos processados como no PCOPS, que apareceu pela primeira vez no artigo que Haessler apresenta a heurística SHP [23]. Entretanto, o SHP não considera em nenhum momento tal formulação.

O método NM apresenta, contudo, uma desvantagem em relação aos outros métodos na maioria das classes testadas: o tempo computacional. Apesar de não podermos comparar com propriedade os tempos computacionais dos métodos, uma vez que não implementamos o método Kombi, é notório na tabela 6.15 que o tempo computacional do NM em algumas classes é alto. Isto é causado principalmente pelo fato de, a cada nova coluna gerada, aplicarmos o BOX9903 em 20 soluções iniciais, objetivando com isso, conforme já observamos, diversificar a busca, tentando "fugir", assim, de mínimos locais. No caso de problemas com grande número de itens, como nas classes (11,12,17,18), o tempo computacional do NM foi alto.

Pode-se aprimorar o desempenho do NM ao tentar outros algoritmos para resolução dos problemas de programação não-linear P_k, bem como outros métodos de arredondamento. Como já dissemos, usamos o método $BRURED$ para arredondar a solução, e este é um dos mais simples métodos de arredondamento.

Outra melhoria que pode diminuir o tempo computacional é tentar outras formas de diversificar a busca pelo ótimo global na resolução do P_k. De fato, a etapa mais custosa do ponto de vista computacional na nossa implementação é resolver o P_k 20 vezes com o BOX9903 a cada iteração, pois usamos 20 soluções iniciais visando "fugir" de ótimos locais.

Também pretendemos linearizar a função objetivo do problema P_k, para assim não só obter os multiplicadores simplex sem precisar recorrer ao PL Auxiliar como para encontrar o ótimo global do problema corrente a cada nova coluna adicionada.

Uma possível melhoria em relação ao número de objetos processados pode ser obtida através da limitação do acréscimo deste quando resolvemos a sequência de problemas P_k. Para isso, basta acrescentar uma restrição do tipo $\sum_{j=1}^{n} x_j \leq NO_0$, onde NO_0 é o número de objetos obtido pela solução inicial.

Por fim, vale observar que o método NM é promissor tanto do ponto de vista teórico como prático no contexto dos Problemas de Corte Unidimensional.

Apêndice A

Classes $\mathcal{P}$ e $\mathcal{NP}$

Dadas duas funções $f, g : \mathbb{N} \rightarrow \mathbb{N}$, dizemos que $f(n)$ é $O(g(n))$ se existem constantes c, a e n_0 tal que, para todo $n \geq n_0$,

$$f(n) \leq cg(n) + a$$

Um problema P é chamado de problema de decisão se todos os exemplos de P pertencem à S_P, conjunto de exemplos positivos, ou pertencem à N_P, conjunto de exemplos negativos. Assim, se X é um exemplo do problema P, então $X \in S_P$ ou $X \in N_P$. Assumimos que qualquer algoritmo para um problema de decisão retorna uma resposta *SIM*, se o exemplo pertence a S_P, ou $N\tilde{A}O$, se o exemplo pertence a N_P.

Definimos o *tamanho* da entrada $L = L(X)$ de um exemplo do problema P, como sendo o número de dígitos (possivelmente bits) necessários para especificar este exemplo do problema P. Denotaremos o tamanho da entrada $L = L(X)$ como sendo $|X|$.

Definição . Dado um problema P, um algoritmo A para o problema, e um exemplo X, seja $f_A(X)$ o número de cálculos elementares necessários para o algoritmo A resolver

o problema para o exemplo X. $f_A^*(l) = \sup_X\{f_A(X) : |X| = l\}$ é o tempo de execução do algoritmo A. Um algoritmo A é polinomial para um problema P se $f_A^*(l) = O(l^p)$ para algum p inteiro positivo.

Definição . $\mathcal{NP}$ é a classe de problemas de decisão com a seguinte propriedade: para qualquer exemplo para o qual a resposta é *SIM*, existe uma "curta" (polinomial) prova do *SIM*.

Definição . $\mathcal{P}$ é a classe de problemas de decisão em $\mathcal{NP}$ para a qual existe um algoritmo polinomial.

Definição . Se $P, Q \in \mathcal{NP}$, e se um exemplo de P pode ser convertido em tempo polinomial num exemplo de Q, então P é polinomialmente redutível a Q.

Definição . $\mathcal{NPC}$, a classe dos problemas $\mathcal{NP}$-completo, é o subconjunto de problemas $P \in \mathcal{NP}$ tal que, para todo $Q \in \mathcal{NP}$, Q é polinomialmente redutível a P.

Proposição A.1. *Suponha que os problemas $P, Q \in \mathcal{NP}$.*

1. *Se $Q \in \mathcal{P}$ e P é polinomialmente redutível a Q, então $P \in \mathcal{P}$.*
2. *Se $P \in \mathcal{NPC}$ e P é polinomialmente redutível a Q, então $Q \in \mathcal{NPC}$.*

Demonstração: (2) Considere qualquer problema $R \in \mathcal{NP}$. Como $P \in \mathcal{NPC}$, R é polinomialmente redutível a P. Como P é polinomialmente redutível a Q por hipótese, temos que R é polinomialmente redutível a Q. Como isto vale para todo $R \in \mathcal{NP}$, $Q \in \mathcal{NPC}$.

Corolário A.1.1. *Se $\mathcal{P} \cap \mathcal{NPC} \neq \emptyset$, então $P = NP$.*

Demonstração: Suponha que $Q \in \mathcal{P} \cap \mathcal{NPC}$ e tome $R \in \mathcal{NP}$. Por (2), como $R \in \mathcal{NP}$ e $Q \in \mathcal{NPC}$, R é polinomialmente redutível a Q. Por (1), como $Q \in \mathcal{NP}$ e R é

polinomialmente redutível a Q, temos que $R \in \mathcal{P}$. Logo $\mathcal{NP} \subset \mathcal{P}$, o que implica $\mathcal{P}=$ $\mathcal{NP}$.

Vale destacar que demonstrar que $\mathcal{P}=\mathcal{NP}$ ou $\mathcal{P}\neq\mathcal{NP}$ é um problema em aberto.

Já podemos agora voltar nossa atenção aos problemas de otimização. O estudo do custo para resolver problemas de otimização é um dos aspectos mais relevantes da teoria de complexidade, visto a importância e as aplicações de tais problemas em várias áreas.

Para aplicarmos a teoria de complexidade computacional que descrevemos acima, devemos associar a cada problema de otimização P um problema de decisão P_D. No caso de P ser um problema de minimização, P_D deve "perguntar", para algum $K > 0$, se existe uma solução viável para o exemplo X com função objetivo menor ou igual a K. Analogamente, se P é um problema de maximização, o problema de decisão associado deve "perguntar" se, dado $K > 0$, existe uma solução viável para o exemplo X cuja função objetivo seja maior ou igual a K.

Mais especificamente, consideremos o seguinte o problema de otimização:

$$\max\{cx : \ x \in S\}$$

onde $X = \{c$ e uma representação "padrão" de S$\}$ consiste em um exemplo deste problema.

Para qualquer inteiro k, associamos ao exemplo X um problema de decisão:

$$\textit{Existe } x \in S \textit{ tal que } cx \geq k?$$

Definição . Um problema de otimização para o qual o problema de decisão está em $\mathcal{NPC}$ é chamado *$\mathcal{NP}$-difícil.*

Martello e Toth [38] demonstram que o Problema da Mochila 0-1 e o Problema da Mochila Limitada (com n itens) são NP-difícil. Consequentemente, não podem ser resolvidos em um tempo limitado por um polinômio em n, a não ser que $\mathcal{P}=\mathcal{NP}$. Porém, eles podem ser resolvidos por algoritmos pseudo-polinomiais, isto é, algoritmos cuja complexidade no tempo (e espaço) é limitado por um polinômio em n e c (capacidade da mochila). Quando não existe tal algoritmo pseudo-polinomial dizemos que o problema é *$\mathcal{NP}$-difícil no sentido forte.* Martello e Toth [38] demonstram que:

1. O Problema de Múltiplas Mochilas 0-1 é *$\mathcal{NP}$-difícil no sentido forte*;
2. O Problema de Empacotamento de *Bin's* é *$\mathcal{NP}$-difícil no sentido forte.*

Consideremos, por fim, o seguinte problema de corte de estoque proposto por McDiarmind [40]:

Minimização de Padrões (*Pattern Minimisation*)

Entrada: naturais $d_1, ..., d_n$.

Tarefa: no problema de corte de estoque, onde a demanda é d_i para o item do tipo i, com a condição de quaisquer dois itens poderem ser colocados na bobina mestre, mas não três, encontrar a solução que minimiza o desperdício e o número de diferentes padrões a serem usados.

McDiarmind demonstrou o seguinte resultado:

Teorema A.2 (McDiarmind). *O problema de Minimização de Padrões é $\mathcal{NP}$-difícil no sentido forte.*

Este teorema reforça a dificuldade que encontramos para resolver o problema de corte para minimizar o número de objetos processados e o setup.

Recomendamos as referências [3, 19, 38, 40, 57], utilizadas para escrever este apêndice, para mais informações e conceitos sobre complexidade computacional e problemas de otimização combinatória.

Apêndice B

Pseudo-Código *Branch-and-bound*

Variáveis de entrada :

- y : variáveis duais vindas da solução ótima obtida do problema relaxado
- Largura : largura da matéria-prima
- w : largura dos itens a serem cortados
- m : número total de itens a serem cortados

Variáveis de saída :

- ValorOtimo : Valor ótimo da função objetivo do problema da mochila
- x : solução parcial do problema da mochila
- coluna : representa a nova coluna a ser gerada, i.é.,solução ótima (x-otimo) do problema da mochila

(Inicialização)

ValorOtimo = 0

k = 0

nmit = 1000 (número máximo de iterações)

nit = 0

(Achando a extensão mais promissora do ramo atual)

Enquanto nit $\leq$ nmit faça

Início

    nit = nit + 1

    para j = (k+1) até (m) faça

    Início

        soma = 0

        para i = (1) até (j-1) faça

        Início

            soma = soma + w(i)*x(i)

        Fim

        x(j) = int((Largura-soma)/w(j))

    Fim

    k = m

    valor = 0

    para i = (1) até (m) faça

    Início

        valor = valor + y(i)*x(i)

    Fim

(Temos uma solução melhor ?)

    se valor > ValorOtimo então

    Início

        ValorOtimo = valor

```
        para j = (1) até (m)
        Início
           coluna( j ) = x( j )
        Fim
    Fim
( Backtracking para o próximo ramo)
10  se k=1 então Fim do Algoritmo
    k = k - 1
    se x(k)=0 vá para linha 10
    x(k)=x(k)-1
( Vale a pena explorar este ramo ?)
    soma1 = 0
    soma2 = 0
    para i = (1) até (k)
    Início
       soma1 = soma1 + y(i)*x(i)
       soma2 = soma2 + w(i)*x(i)
    Fim
    valor = soma1 + y(k+1)*(Largura-soma2)/w(k+1)
    se valor ≤ (ValorOtimo) vá para linha 10
Fim
Fim
```

Apêndice C

Pseudo-Código do Gerador de Soluções Iniciais para P_k

Observação: Este pseudo-código foi utilizado por Martinez em startcos.for [39, 58].

Variáveis de Entrada:

- n - número de variáveis
- isem - parâmetro inicial (inteiro)

Variáveis de Saída: x(j) inteiro não-negativo , j=1,...,n - frequências geradas

```
isem=237
para i = (1) até (n) faça
Início
        x(i) = rnd(isem)*10.d0
Fim
```

Função rnd(sem)

Variáveis Inteiras: sem, mult

Início

sem = mod(mult(sem, 3141581) + 1, 100000000)

rnd = sem/100000000.0d0

Return

FIM

Função mult(p, q)

Variáveis Inteiras: p, q, p0, p1, q0, q1

Início

p1 = p/10000

p0 = mod(p,10000)

q1 = q/10000

q0 = mod(q,10000)

mult = mod(mod(p0*q1+p1*q0,10000)*10000+p0*q0,100000000)

Return

Fim

Bibliografia

[1] ALLWOOD, J. M., and GOULIMINS, C. N., *"Reducing the number of patterns in one-dimensional cutting stock problems"*, Control Section Report No EE/CON/IC/88/10, Industrial Systems Group, Department of Electical Engineering, Imperial College, London, 1988.

[2] ARENALES, M. N., MORABITO, R. e YANASSE, H. H., *"O Problema de Corte e Empacotamento e Aplicações Industriais"*, XX CNMAC, Gramado - RS, 1997.

[3] AUSIELLO, G., CRESCENZI, P., GAMBOSI, G., KANN, V., MARCHETTI-SPACCAMELA, A., PROTASI, M., *"Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties"*, Springer-Verlag, Berlim-Heidelberg-Nova York, 2003.

[4] BARNHART, C., JOHNSON, E. L., NEMHAUSER, G. L., SAVELSBERGH, M. W. P. e VANCE, P. H., *"Branch-and-Price: Column Generation for Solving Huge Integer Programs"*, Mathematical Programming, State of Art, livro do International Symposium on Mathematical Programming, 1994.

[5] BELOV, G. e SCHEITHAUER, G., *"The number of setups (different patterns) in one-dimensional stock cutting"*, Technical Report, Dresden University, 2003.

[6] BISCHOFF, E. e RATCLIFF, M. S., *"Loading Multiple Pallets"*, Journal of the Operational Research Society, 46, pp.1322-1336, 1995.

[7] BISCHOFF, E. e WASCHER, G. *"Cutting and packing"*, European Journal of Operational Research, 84, n. 3, pp. 1-5, 1995.

[8] CHVATAL, Vasek, *"Linear Programming"*, Freeman, 1983.

[9] CONN, A. R. e MONGEAU, M., *"Discontinuous piecewise linear optimization"*, Mathematical Programming, 90, pp. 315-380, 1998.

[10] CORREIA, M. H., OLIVEIRA, J. P. e FERREIRA, J. S., *"Reel and Sheet Cutting at a Paper Mill"*, Computers & Operations Research, 31, pp. 1223-1243, 2004.

[11] DIEGEL, A., CHETTY, M., VAN SCHALKWYCK S., e NAIDOO, S., *"Setup combining in the trim loss problem - 3-to-2 & 2-to-1"*, Working paper, Business Administration, University of Natal, Durban, First Draft, 1993.

[12] DIEGEL, A., MONTOCCHIO, E., WALTERS, E., SCHALKWYK, S. e NAIDOO, S., *"Setup minimising conditions in the trim loss problem"*, European Journal of Operational Research, 95, pp.631-640, 1996.

[13] DOWSLAND, K. e DOWSLAND, W., *"Packing Problems"*, European Journal of Operational Research, 56, pp. 2-14, 1992.

[14] DYCKHOFF, H., *"A typology of cutting and packing problems"*, European Journal of Operation Research, 444, pp. 145-159, 1990.

[15] DYCKHOFF, H. e FINKE, U., *"Cutting and Packing in Production and Distribution: A Typology and Bibliography"*, Springer-Verlag Co, Heidelberg, 1992.

[16] EILON, S., *"Optimizing the shearing of steel bars"*, Journal of Mechanical Engineering Science, 2, pp. 129-142, 1960.

[17] EILON, S. e CHRISTOFILDES, N., *"The Loading Problem"*, Managemente Science, 17, pp. 256-268, 1971.

[18] FOESTER, H. e WASCHER, G., *"Pattern Reduction in One-dimensional Cutting-Stock Problems"*, International Journal of Prod. Res., 38, pp. 1657-1676, 2000.

[19] GAREY, M. R. e JOHNSON, D.S., *"Computers and Intractability"*, Freeman, São Francisco, 1979.

[20] GAU, T. e WASCHER, G., *"CUTGEN1: A Problem Generator for the Standard One-dimensional Cutting Stock Problem"*, European Journal of Operational Research, 84, pp. 572-579, 1995.

[21] GILMORE, P.C. e GOMORY, R.E., *"A Linear Programming Approach to the Cutting Stock Problem I"*, Operations Research, 9, pp. 849-859, 1961.

[22] GILMORE, P.C. e GOMORY, R.E., *"A Linear Programming Approach to the Cutting Stock Problem II"*, Operations Research, 11, pp. 863-888, 1963.

[23] HAESSLER, R., *"Controlling Cutting Pattern Changes in One-Dimensional Trim Problems"*,Operations Research, 23, pp. 483-493, 1975.

[24] HAESSLER, R., *"A Note on Computational Modifications to the Gilmore-Gomory Cutting Stock Algorithm"*, Operations Research, 28, pp. 1001-1005, 1980.

[25] HARDLEY, C. J., *"Optimal cutting of zinc-coated steel strip"*, Operational Research, 4, pp. 92-100, 1976.

[26] HINXMAN, A., *"The trim loss and assortment problems: a survey"*, European Journal of Operational Research, 5, pp 8-18, 1980.

[27] HODGSON, T., *"A Combined Approach to the Pallet Loading Problem"*, IIE Transactions 14, pp 176-182, 1982.

[28] HOROWITZ, E. e SAHNI, S. *"Computing partitions with applications to the Knapsack Problem"*, Journal of ACM, 21, pp 277-292, 1974.

[29] KAMIMURA, K. e NISHINO, H., *"An efficient method for determining economical configurations of elementary parcket-switched networks"*, IEEE Transactions on Communications 39, pp. 278-288, 1991.

[30] KANTOROVICH, L. V., *"Mathematical Methods of Organizing and Planning Production"*, Management Science., 6, pp 366-422, 1960.

[31] KOLESAR, P. J., *"A branch and bound algorithm for the knapsack problem"*, Management Science., 13, 723-735, 1967.

[32] KREJIC, N., MARTINEZ, J. M., MELLO, M. P. e PILOTTA, E. A., *"Validation of an Augmented Lagrangian algorithm with a Gauss-Newton Hessian approximation using a set of hard-spheres problems"*, Computational Optimization and Applications 16, pp. 247-263, 2000.

[33] JOHNSTON, R. E., *"Rounding algorithms for cutting stock problems"*, Asia-Pacific Journal of Operational Research, 3, pp. 166-171, 1986.

[34] LIMEIRA, M. S., *"Redução do número mínimo de padrões em problemas de corte de estoque"*, Tese de Doutorado apresentada no Instituto Nacional de Pesquisas Espaciais - INPE - São José dos Campos , 2003.

[35] MAIA LOA, VALÉRIO DE CARVALHO, L. A. e QUASSIM, R. Y., *"Synthesis of utility systems by simulated annealing"*, Computers and Chemical Engineering, 19, pp. 481-488, 1995.

[36] MARTELLO, S. e TOTH, P., *"An Upper Bound fot the Zero-one Knapsack Problem and a branch and bound algorithm"*, European Journal of Operational Research, 1, pp. 169-175, 1977.

[37] MARTELLO, S. e TOTH, P., *"Branch and Bound Algorithms for the Solution of the general unidimensional Knapsack Problem"*, Advances in Operations Research, pp. 295-301, 1977.

[38] MARTELLO, S. e TOTH, P., *"Knapsack Problems: Algotithms and Computer Implementations"*, John Wiley & Sons, 1990.

[39] MARTINEZ, J.M. , *"Minimization of discontinuous cost functions by smoothing"*, Acta Applicandae Mathematical, 71, pp. 245-260, 2001.

[40] McDIAMIRD, C., *"Pattern minimisation in cutting stock problems"*, Discrete Applied Mathematics, 98, pp.121-130, 1999.

[41] METZGER, R. W., *"Stock Slitting"*, Elementary Mathematical Programming, Wiley, 1958.

[42] MORABITO, R. e ARENALES, M. N., *"Um Exame dos Problemas de Corte e Empacotamento"*, Pesquisa Operacional, 12, pp. 1-20, 1992.

[43] MORABITO, R. e ARENALES, M. N., *"An AND/OR-Graph Approach to the Container Loading Problem"*, International Transactions in Operational Research, 1, pp. 59-73, 1994.

[44] PAULL, A. E. e WALTER, J. R., *"The trim problem: an application of linear programming to the manufacture of news-print paper"*, Presented at Annual Meeting of Econometric Society, Montreal, pp. 10-13, 1954.

[45] POLDI, K. C., *"Algumas extensões ao problema de corte de estoque"*, Dissertação de Mestrado apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMS-USP, São Carlos, 2003.

[46] STADLER, H., *"A one-dimensional cutting stock problem in the aluminium industry and its solution*, European Journal of Operational Research, 44, pp. 209-223, 1990.

[47] SHULMAN, A. e VECHANI, R., *"A decomposition algorithm for capacity expansion of local access networks"*, IEEE Transaction on Communications, 41, pp. 1063-1073, 1993.

[48] TEGHEM, J., PIRLOT, M. e ANOTONIADIS, C., *"Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem"*, Journal of Computational and Applied Mathematics, 64, pp. 91-102, 1995.

[49] UMETAMI, S. e YAGIURA, M. e IBARAKI, T., *"One Dimensional Cutting Stock Problem to Minimize the Number of Different Patterns"*, European Journal of Operational Research, vol.146, No.2, pp.388-402, 2003.

[50] VALÉRIO de CARVALHO, J. M., *"Exact solution os cutting stock problems using column generation and branch-and-bound"*, International Transactions in Operational Research, 5, pp. 35-44, 1998.

[51] VANDERBECK, F., *"On Dantizg-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm"*, Research Papers in Management Studies, University of Cambridge, 1994-1995, no 29.

[52] VANDERBECK, F., *"Computational Study of a Column Generation Algorithm for Bin Packing and Cutting Stock Problems"*, Mathematical Programming, 86, no 3, pp. 565-594, 1999.

[53] VANDERBECK, F., *"Exact Algorithm for Minimizing the Number of Setups in the one-dimensional cutting stock problem"*, Operations Research, 48, pp. 915-926, 2000.

[54] VANDERBECK, F. e WOLSEY, A., *"An Exact Algorithm for IP Column Generation"*, Operations Research Letters, 19, pp. 151-159, 1996.

[55] WALKER, J., *"Decision support for the single-period inventory problem"*, Industrial Management & Data Systems, 100, pp. 61-66, 2000.

[56] WASCHER, G. e GAU, T., *"Heuristics for the Integer One-dimensional Cutting Stock Problem: a computational study"*, OR Spektrum, 18, pp. 131-144, 1996.

[57] WOLSEY, L., *"Integer Programming"*, John Wiley & Sons, Nova York, 1998.

[58] `http://www.ime.unicamp.br/~martinez/software.htm`.

Índice