Universidade Estadual de Campinas

Instituto de Matemática Estatística e Computação Científica

Departamento de Matemática Aplicada

Uma formulação não-linear para o problema de corte unidimensional

Momoe Sakamori*

Mestrado em Matemática Aplicada - Campinas - SP

Orientador: Profa. Dra. Márcia A. Gomes Ruggiero

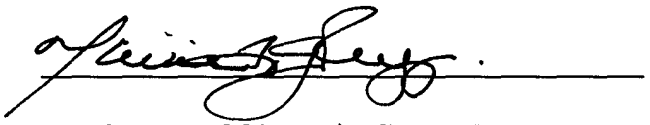
Co-Orientador: Prof. Dr. Antonio Carlos Moretti

∗ Este trabalho teve apoio financeiro da CAPES.

Uma formulação não-linear para o problema de corte unidimensional

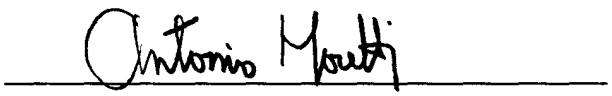
Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por **Momoe Sakamori** e aprovada pela comissão julgadora.

Campinas, 04 de Julho de 2008.



Profa. Dra. Márcia A. Gomes Ruggiero

Orientadora



Prof. Dr. Antonio Carlos Moretti

Co-Orientador

Banca Examinadora

Profa. Dra. Márcia A. Gomes Ruggiero (IMECC - UNICAMP)

Prof. Dr. Luiz Leduíno de Salles Neto (UNIFESP)

Profa. Dra. Valéria Abrão de Podestá (IMECC - UNICAMP)

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica, Unicamp, para obtenção do Título de MESTRE em Matemática Aplicada.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues

Sakamori, Momoe
Sa29f Uma formulação não-linear para o problema de corte unidimensional / Momoe Sakamori -- Campinas, [S.P. :s.n.], 2008.

Orientadores : Márcia Aparecida Gomes Ruggiero ; Antonio Carlos Moretti
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Problema do corte de estoque. 2. Heurística. 3. Programação não-linear. . I. Ruggiero, Márcia Aparecida Gomes. II. Moretti, Antonio Carlos. III. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. IV. Título.

Título em inglês: A nonlinear formulation for the unidimensional cutting-stock problem.

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

Área de concentração: Pesquisa operacional

Titulação: Mestre em Matemática Aplicada

Banca examinadora:

Profa. Dra. Márcia Aparecida Gomes Ruggiero (IMECC-UNICAMP)
Prof. Dr. Luiz Leduino de Salles Neto (UNIFESP)
Profa. Dra. Valéria Abrão de Podestá (IMECC-UNICAMP)

Data da defesa: 04/07/2008

Programa de pós-graduação: Mestrado em Matemática Aplicada

Dissertação de Mestrado defendida em 04 de julho de 2008 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.

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

Prof. (a). Dr (a). LUIZ LEDUINO SALLES NETO

Prof. (a). Dr (a). VALÉRIA ABRÃO DE PODESTÁ

Agradecimentos

Agradeço:

Aos meus pais, por me incentivarem e possibilitarem os meus estudos.

À professora Márcia, minha "mãe acadêmica"desde a graduação, pela orientação, apoio e amizade.

Ao professor Moretti, pela co-orientação do trabalho.

Aos membros da banca, pelas sugestões e correções da dissertação.

Aos funcionários do IMECC, que sempre me ajudaram quando precisei.

Às minhas irmãs, que sempre estiveram ao meu lado incentivando.

Ao Feodor, pela ajuda, apoio e carinho.

Aos meus amigos, pela torcida e incentivo.

À CAPES pelo apoio finaceiro.

Resumo

Neste trabalho resolvemos um problema de corte unidimensional não-linear para minimizar o número de objetos processados, *setup* e desperdício. O termo não-linear representa o *setup* da máquina de corte. Resolvemos o problema utilizando o pacote MINOS e obtemos a solução inteira através de um procedimento heurístico. Como o número de padrões de corte pode ser muito grande, propomos uma geração de colunas modificada, que usa os multiplicadores de Lagrange do problema não-linear ao invés das variáveis duais do problema de programação linear padrão. Além disso, propomos um novo processo de geração de colunas utilizando um problema da mochila não-linear como subproblema para gerar colunas promissoras.

Palavras-chave: problema de corte; geração de coluna; *setup*; programação não-linear.

Abstract

In this work we solve a nonlinear unidimensional cutting-stock problem to minimize the number of objects processed, setup and trim loss. The nonlinear term represents the setup of the cutting machine. We solve the problem using the MINOS package and obtain the integer solution through a heuristic procedure. Since the number of cutting patterns can be very high, we propose a modified column generation that uses the Lagrange multipliers of the nonlinear problem instead the dual variables of the standard linear programming problem. Also, we propose a new column generation process using a nonlinear knapsack problem as the subproblem to generate profitable columns.

Keywords: cutting-stock problem; column generation; setup; nonlinear programming.

Sumário

Lista de Algoritmos

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os problemas de corte são amplamente utilizados nas indústrias, demonstrando a importância de seu estudo. A formulação mais comum é a de um problema de programação linear para minimizar o número de objetos processados. Na realidade, há outros fatores que podem ser considerados na resolução dos problemas corte, como por exemplo, o custo de troca de um padrão para o outro, conhecido como *setup*. Em 1975, Haessler [14] propôs uma heurística abordando este custo. Em 2005, Salles Neto [20] utilizou uma formulação não-linear para o problema de corte unidimensional com o objetivo de minimizar o número de objetos processados e o *setup*. Para este modelo não-linear, formulou-se um problema auxiliar de programação linear para a obtenção das variáveis duais necessárias para a geração de colunas no método de geração de colunas de Gilmore e Gomory [9, 10]. Há outros trabalhos que também abordam o custo de *setup*, [3, 21, 22, 23, 25].

Baseado no trabalho de Salles Neto [20], o nosso objetivo neste trabalho é resolver um problema de corte unidimensional considerando como objetivos minimizar além do número de objetos processados e o *setup*, o desperdício. Estes objetivos são conflitantes e não encontramos nenhum trabalho em que os resultados dos problemas de CUTGEN1 [8] tenham sido indicados nestes três aspectos. Umetani, Yagiura e Ibaraki, [21, 22], apresentaram os resultados de seu método ILS-APG apenas em relação ao *setup* e ao desperdício, e da modificação deste em relação ao *setup* e ao número de objetos processados. Para a fase de geração de padrões, propomos utilizar os multiplicadores de Lagrange da solução do problema não-linear ao invés das variáveis duais da solução do problema auxiliar de programação linear. Além disso, propomos uma formulação não-linear para a geração

de colunas. Estudamos a heurística *Simulated Annealing* como uma estratégia de globalização para a resolução do problema não-linear. Para a resolução de problema não-linear, utilizamos o pacote de otimização MINOS [18]. As mudanças e adaptações feitas neste trabalho com relação a [20], foram realizadas com o intuito de obter um método mais eficiente, tanto do ponto de vista da solução obtida como do tempo de execução.

Este texto está organizado da seguinte maneira. No Capítulo 2 apresentamos a formulação de um problema de corte unidimensional, uma formulação como problema de programação não-linear e uma estratégia de globalização para a sua resolução. Além disso, apresentamos alguns métodos de geração de colunas, heurísticas de arredondamento da solução e uma proposta para o custo de *setup*. As implementações e testes numéricos são descritos e analisados no Capítulo 3. O texto é concluído com algumas considerações finais no Capítulo 4.

Ao longo do texto, $\lfloor x \rfloor$ é o maior inteiro menor ou igual a x e $\lceil x \rceil$ é o menor inteiro maior ou igual a x.

Capítulo 2

Formulação

2.1 Formulação

Os problemas de corte são de grande interesse devido a suas aplicações nas indústrias de papel, de aço, moveleiras, têxtil, entre outras. Em um problema de corte, peças de larguras pré-especificadas devem ser cortadas de uma placa, ou rolo no caso unidimensional, de modo a atender a uma demanda estabelecida. Conhecidas as larguras e demandas de cada item, diferentes padrões de corte podem ser montados, que caracterizam o problema como um problema combinatorial. Além de atender à demanda, o objetivo do problema é como melhor cortar as peças de forma a obter, por exemplo, o menor número de rolos processados, o menor desperdício ou o menor número de diferentes padrões de corte.

A seguir detalhamos o problema para o caso unidimensional.

2.1.1 Problema de corte unidimensional

O problema de corte unidimensional consiste em cortar objetos (rolos) de mesma largura W em m diferentes itens (peças) de larguras $w_1, w_2, \ldots, w_m$, com respectivas demandas $d_1, d_2, \ldots, d_m$. Um padrão de corte j para este problema é definido por um vetor a_j, cuja componente a_{ij} representa o número de vezes que o item i será cortado no tamanho w_i, e satisfazem a condição:

$$\sum_{i=1}^{m} w_i a_{ij} \leq W, \quad \forall j.$$

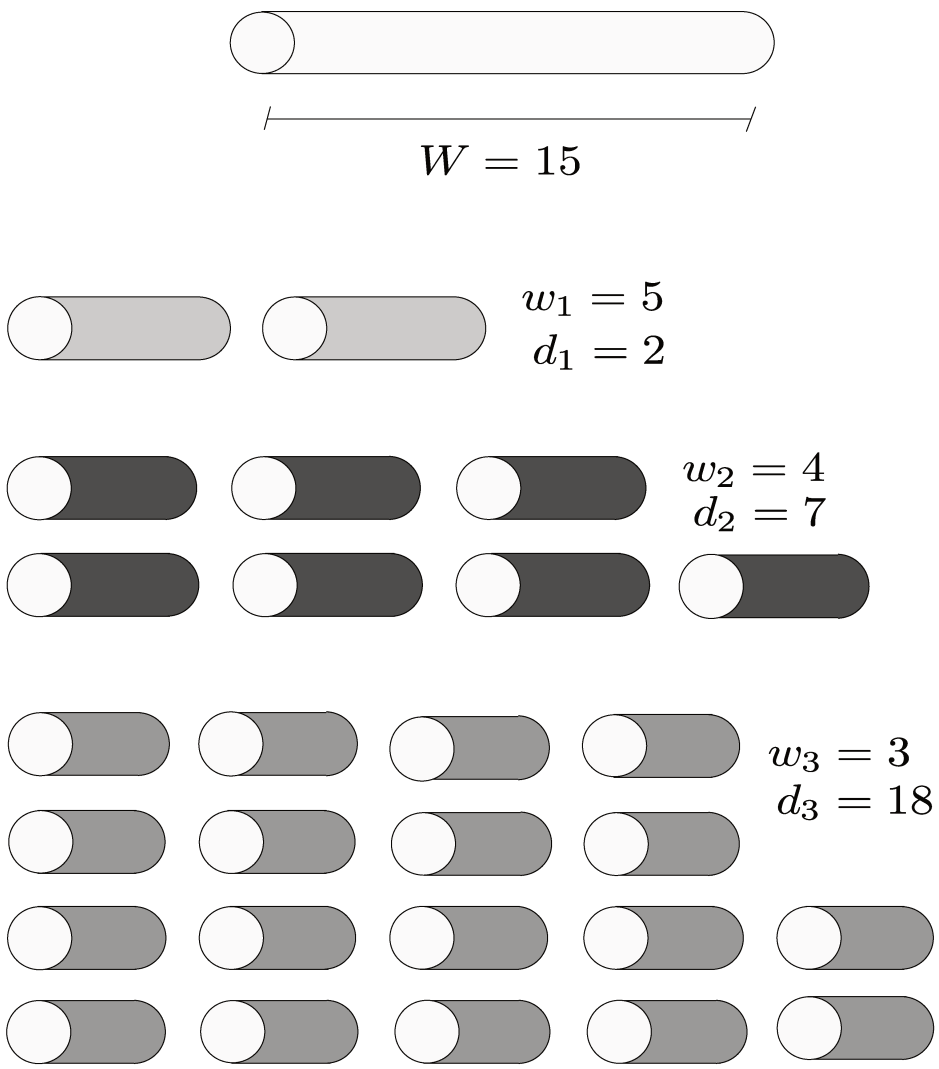


Figura 2.1: Exemplo.

Supondo que temos objetos de tamanho $W = 15$ e queremos cortá-los de forma a obter 3 itens diferentes ($m = 3$): 2 itens de tamanho 5 ($w_1 = 5$, $d_1 = 2$), 7 itens de tamanho 4 ($w_2 = 4$, $d_2 = 7$) e 18 itens de tamanho 3 ($w_3 = 3$, $d_3 = 18$). A Figura 2.1 ilustra os dados do problema.

Um possível padrão de corte é cortar a rolo mestre de forma a obter 3 itens de tamanho 4 e 1 item de tamanho 3. Representamos este primeiro padrão por $a_1 = (0, 3, 1)^T$. Observe que neste caso não temos nenhuma perda com o padrão, isto é, não houve sobras, como mostra a Figura 2.2.

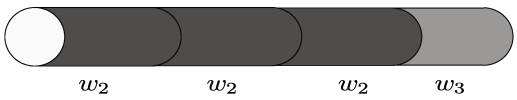


Figura 2.2: Exemplo de padrão sem perda.

Nem sempre é possível construir padrões sem perda, conforme mostra a Figura 2.3. Com este padrão obtemos 1 item de tamanho 5, 2 itens de tamanho 4 e há uma perda de 2. Este segundo padrão é representado como $a_2 = (1, 2, 0)^T$.

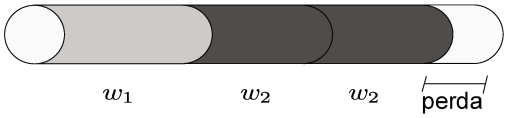


Figura 2.3: Exemplo de padrão com perda.

Um outro tipo de padrão que podemos construir são os chamados padrões homogêneos, em que é escolhido apenas um tipo de item do qual corta-se o maior número de vezes possível. Neste exemplo, é possível cortar 3 itens de tamanho 5, ou 3 itens de tamanho 4 ou ainda 5 itens de tamanho 3, gerando os padrões homogêneos $a_3 = (3, 0, 0)^T$, $a_4 = (0, 3, 0)^T$ e $a_5 = (0, 0, 5)^T$, como mostra a Figura 2.4.

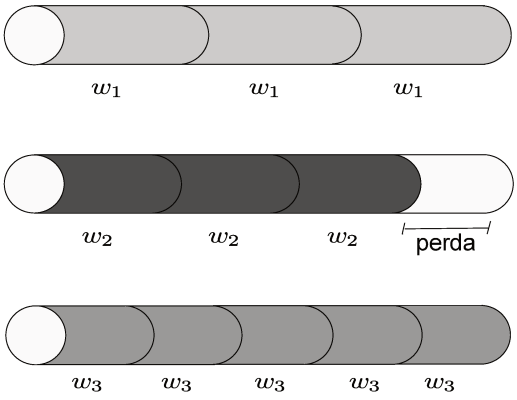


Figura 2.4: Exemplos de padrões homogêneos.

Uma das dificuldades do problema é gerar todos os possíveis padrões de corte, dado que há um número muito grande deles. Considerando um conjunto de n padrões de corte previamente definidos, formulamos o modelo do problema de corte unidimensional como um problema de programação linear inteira com o objetivo de minimizar o número de

objetos processados, assim denominado pois a cada padrão cortado corresponde uma peça do objeto de tamanho W:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{j=1}^{n} x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j = d_i, \quad i = 1, \ldots, m \\ & x_j \geq 0 \text{ e inteiro}, \quad j = 1, \ldots, n, \end{aligned} \tag{2.1}$$

onde cada variável x_j representa a quantidade do padrão de corte do tipo j a ser usado no planejamento de corte. É fácil ver que este problema (com restrições de igualdade) é equivalente ao problema de minimizar o desperdício.

O desperdício com o padrão de corte j é dado por:

$$W - \sum_{i=1}^{m} a_{ij} w_i. \tag{2.2}$$

O desperdício total obtido com todos os padrões é dado por:

$$\sum_{j=1}^{n} \left(W - \sum_{i=1}^{m} a_{ij} w_i \right) x_j. \tag{2.3}$$

Devido às restrições de igualdade, (2.3) pode ser reescrito como:

$$W \sum_{j=1}^{n} x_j - \sum_{i=1}^{m} w_i d_i. \tag{2.4}$$

Como W, w, d são constantes, o problema de minimizar o desperdício é equivalente ao de minimizar o número de objetos processados.

A seguir detalharemos algumas modificações do Problema de programação linear inteira (2.1).

2.1.2 Objetos processados

No Problema (2.1) as restrições relacionadas à demanda são de igualdade. A fim de ampliar o nosso espaço de soluções, trabalhamos com restrições de desigualdade na demanda.

Queremos que a demanda seja atendida, mas relaxamos a condição de que seja exatamente. Isso é justificável, já que nas indústrias há uma certa tolerância na produção dos itens, isto é, aceita-se um excesso na produção. Dessa forma, temos o seguinte problema de minimização do número de objetos processados:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{j=1}^{n} x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \quad i = 1, \ldots, m \\ & x_j \geq 0 \text{ e inteiro}, \quad j = 1, \ldots, n. \end{aligned} \tag{2.5}$$

É importante notar que este problema não é equivalente ao de minimizar o desperdício, pois o desenvolvimento (2.4) depende da suposição que as restrições são de igualdade. A seguir formulamos o problema de minimizar o desperdício relativo, com restrições de desigualdade.

2.1.3 Desperdício relativo

Ao invés de minimizar o número de objetos processados, podemos minimizar as perdas com os padrões. Então, temos o problema de minimizar o desperdício relativo:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{j=1}^{n} t_j x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \quad i = 1, \ldots, m \\ & x_j \geq 0 \text{ e inteiro}, \quad j = 1, \ldots, n, \end{aligned} \tag{2.6}$$

onde a razão do desperdício do padrão j pela largura total W do objeto é dada por:

$$t_j = \frac{W - \sum_{i=1}^{m} a_{ij} w_i}{W}. \tag{2.7}$$

2.1.4 *Setup*

Na formulação do problema de corte unidimensional, a função objetivo pode incorporar o custo de *setup*. Este termo se refere ao custo de troca de padrão, isto é, a cada padrão novo a ser utilizado é necessário fazer um ajuste e reposicionamento das facas de corte das máquinas, o que exige tempo e trabalho.

Incorporando o custo de *setup* ao problema (2.5) teremos o modelo:

$$\begin{aligned} \text{Minimizar} \quad & c_1 \sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \delta(x_j) \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \qquad i = 1, \ldots, m \\ & x_j \geq 0 \text{ e inteiro}, \qquad j = 1, \ldots, n, \end{aligned} \tag{2.8}$$

onde c_1 é o custo de cada objeto processado e c_2 é o custo de *setup* relativo a cada troca de padrão de corte, e,

$$\delta(x_j) = \begin{cases} 0, \text{ se } x_j = 0 \\ \\ 1, \text{ caso contrário.} \end{cases}$$

Dessa forma, temos que cada padrão incorporado na solução tem um custo de c_2 na função objetivo.

2.1.5 Problema de corte unidimensional considerando o número de objetos, *setup* e desperdício relativo

Além de minimizar o número de objetos processados e o *setup*, consideramos a minimização do desperdício relativo. Introduzimos mais um termo na função objetivo, e chamaremos de problema de minimização do número de objetos processados, *setup* e desperdício relativo (POSD):

$$\begin{aligned} \text{Minimizar} \quad & c_1 \sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \delta(x_j) + c_3 \sum_{j=1}^{n} t_j x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \qquad i = 1, \ldots, m \\ & x_j \geq 0 \text{ e inteiro}, \qquad j = 1, \ldots, n. \end{aligned} \tag{2.9}$$

Este problema pode ser reduzido aos Problemas (2.5), (2.6), (2.8), ajustando-se os pesos c_1, c_2 e c_3.

2.2 Esquema de resolução

Conforme destacamos na Seção 2.1, dada a natureza combinatória do problema, é inviável gerar todos os possíveis padrões de corte. Desta forma, o procedimento de resolução deve incluir uma etapa na qual um novo padrão de corte é gerado e incluído ao conjunto de padrões já estabelecido. Além disso, o problema é resolvido com a relaxação da restrição de integralidade das variáveis. Então, ao final da resolução um processo de arredondamento deve ser executado.

As etapas principais da resolução de um problema de corte são:

- obtenção de uma solução inicial: conjunto inicial de padrões de corte;
- resolução do problema para o conjunto de padrões definidos até o momento;
- geração de um novo padrão (coluna);
- arredondamento da solução.

A Figura 2.5 mostra o fluxograma da resolução de um problema de corte. O critério de parada será explicitado no Capítulo 3.

Nas subseções seguintes serão detalhadas as formas de se realizar cada uma destas etapas.

2.3 Geração de padrões iniciais

Já destacamos que para a maioria dos problemas de corte, gerar todos os padrões de corte possíveis é computacionalmente inviável. Precisamos de um conjunto inicial de padrões para a resolução do problema. A seguir, descrevemos dois destes procedimentos.

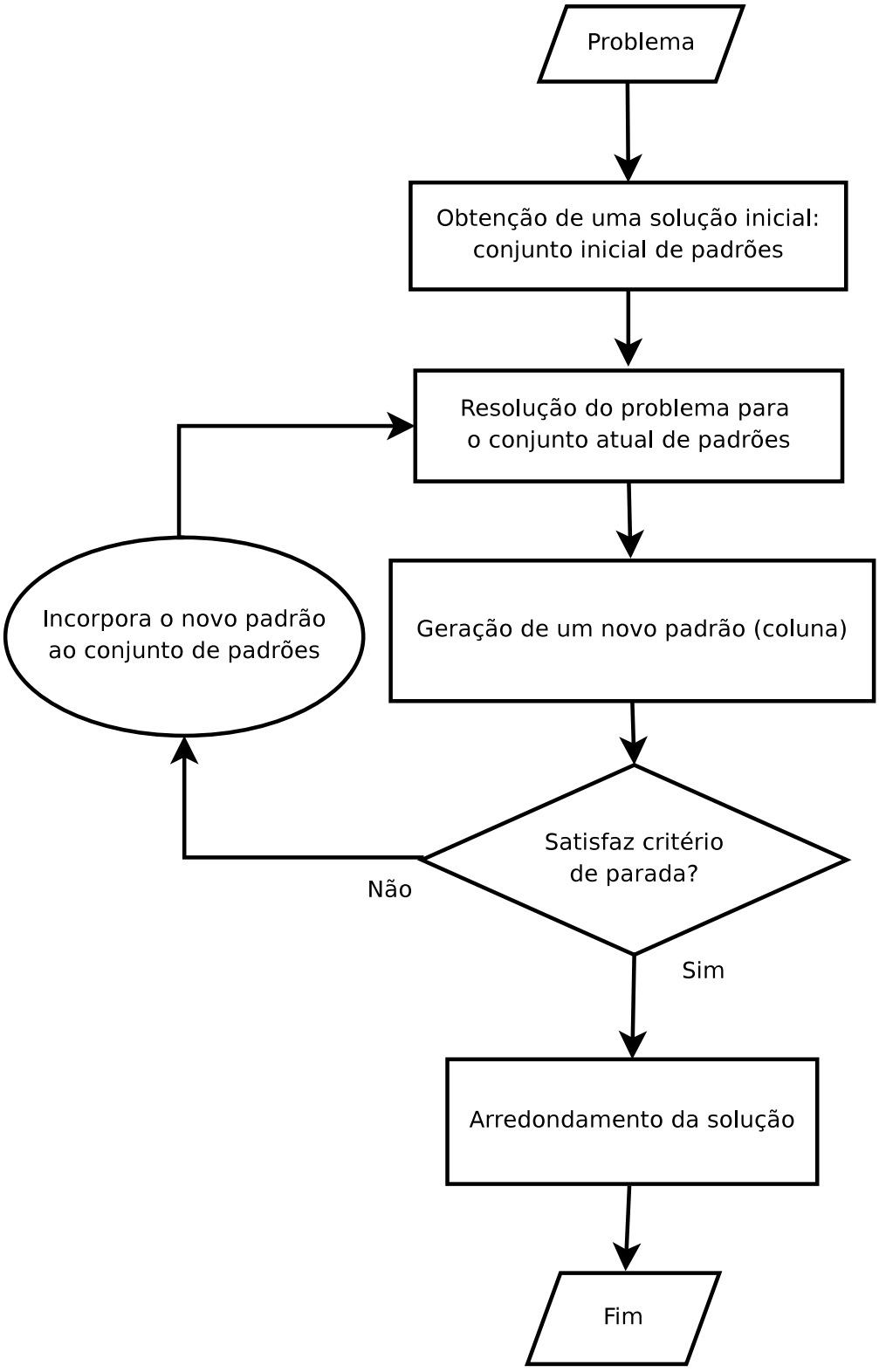


Figura 2.5: Fluxograma da resolução de um problema de corte.

2.3.1 Padrões homogêneos

Podemos construir uma solução inicial utilizando os padrões homogêneos. Um padrão de corte homogêneo é obtido extraindo do objeto um único item, o máximo número de vezes possível. Assim, para cada item, $i = 1, \ldots, m$, o padrão homogêneo é dado por $a_i = (0, \ldots, a_{ii}, \ldots, 0)$, com $a_{ii} = \left\lfloor \frac{W}{w_i} \right\rfloor$. O número de objetos para atender a demanda é dado por $x_i = \left\lceil \frac{d_i}{a_{ii}} \right\rceil$.

2.3.2 SHP - Sequential Heuristic Procedure

Haessler propôs em 1975, [14], a heurística SHP (Sequential Heuristic Procedure), que busca construir padrões de corte visando minimizar o desperdício e o *setup*. É um processo iterativo em que um padrão é gerado, em seguida calcula-se a frequência com que ele é utilizado e a demanda residual que ainda deverá ser satisfeita. Cada padrão gerado a_j procura atender aos seguintes critérios de aspiração:

(i) $W - \sum_{i=1}^{m} a_{ij} w_i \leq MAXTL$

(ii) $MINR \leq \sum_{i=1}^{m} a_{ij} \leq MAXR$

(iii) $x_j \geq MINU$

onde

- $MAXTL$ é a perda máxima permitida com o padrão;
- $\bar{d}_i$ é a demanda residual do item i;
- $NR = \dfrac{\sum_{i=1}^{m} \bar{d}_i w_i}{W}$ é a estimativa do número de objetos para satisfazer a demanda residual;
- $NI = \dfrac{\sum_{i=1}^{m} \bar{d}_i}{NR}$ é o número médio de itens que devem ser obtidos no padrão;
- $MINR = NI - 1$ é o número mínimo de itens no padrão;
- $MAXR$ é o número máximo de itens no padrão (depende da quantidade de facas da máquina de corte);

- $MINU = \alpha * NR$ é o número mínimo de objetos processados com o padrão, α variando entre 0.5 e 0.9.

Os parâmetros $MAXR$, α e $MAXTL \in [0.006W, 0.03W]$, são dados de entrada para a heurística.

Os passos deste procedimento estão discutidos no Algoritmo 1.

A fim de obter melhores resultados, foram feitas algumas modificações na heurística SHP, seguindo a proposta de Salles Netto, [20]. Ao invés de manter constantes os parâmetros $MAXTL$ e $MINU$, como prevê a proposta original, a cada padrão gerado serão atualizados até chegar a um número limite. Iniciamos com $MAXTL = 0.01W$ e $MINU = 0.5NR$. A cada padrão gerado, aumentamos $MAXTL$ em 0.25% de W e $MINU$ em 5% de NR. Os limites superiores para os parâmetros $MAXTL$ e $MINU$ são, respectivamente, $0.03W$ e $0.9NR$.

- Inicial: $MAXTL = 0.01W$

- A cada padrão gerado: $MAXTL = MAXTL + 0.0025W$

- Limite: $MAXTL = 0.03W$

- Inicial: $MINU = 0.5NR$

- A cada padrão gerado: $MINU = MINU + 0.05NR$

- Limite: $MINU = 0.9NR$

O aumento no valor de $MAXTL$ permite a geração de um padrão com maior desperdício, enquanto o aumento no valor de $MINU$ exige uma maior utilização do padrão. Dessa forma, relaxa-se o objetivo de minimizar o desperdício para dar mais ênfase na minimização do *setup*.

Algoritmo 1 SHP - Sequential Heuristic Procedure.

Passo 0: Dados de entrada: W, m, w, d, $MAXR$.

Passo 1: Faça $\bar{d}_i = d_i$, $i = 1, \ldots, m$ e $j = 1$.

Passo 2: Calcule NR , NI, $MINR$, $MINU$, $MAXTL$, $l = 0$.

Passo 3: Ordene os itens por ordem decrescente de demanda residual tal que

$$\bar{d}_1 \geq \bar{d}_2 \geq \ldots \geq \bar{d}_m.$$

Passo 4: Calcule o limitante superior para cada item no padrão

$$b_i = \left\lfloor \frac{\bar{d}_i}{MINU} \right\rfloor, i = 1, \ldots, m.$$

Passo 5: Faça $l = l + 1$ e $a_1^{(l)} = \min\left\{ \left\lfloor \frac{\bar{d}_1}{w_1} \right\rfloor, b_1 \right\}$.

Passo 6: Gere padrão $a^{(l)}$, viável, com $a_1^{(l)}$ fixo, e respeitando os limitantes, isto é,

$$a_i^{(l)} \leq b_i,\ i = 2, \ldots, m$$

Passo 6.1: Testar os critérios de aspiração:

$$\text{(i) } W - \sum_{i=1}^{m} a_i^{(l)} w_i \leq MAXTL;$$

$$\text{(ii) } MINR \leq \sum_{i=1}^{m} a_i^{(l)} \leq MAXR.$$

Se $a^{(l)}$ satisfaz (i) e (ii), faça padrão $a_j = a^{(l)}$ e ir para o Passo 7.

Caso contrário:

Passo 6.2: Se $a_1^{(l)} > 1$, faça $a_1^{(l)} = a_1^{(l)} - 1$, $l = l + 1$ e volte ao Passo 6.

Passo 6.3: Se $a_1^{(l)} = 1$ e $MINU > 1$, faça $MINU = MINU - 1$, $l = l + 1$ e volte ao Passo 4.

Passo 6.4: Se $a_1^{(l)} = 1$ e $MINU = 1$, escolha o padrão com menor perda entre os padrões $a^{(r)}$, $r = 1, \ldots, l$.

Passo 7: Calcule $x_j = \min\left\{ \left\lfloor \frac{\bar{d}_i}{a_{ij}} \right\rfloor, i = 1, \ldots, m \right\}$.

Passo 8: Atualize a demanda residual

$$\bar{d}_i = \bar{d}_i - a_{ij} x_j \quad , \quad i = 1, \ldots, m.$$

Passo 8.1: Se $\bar{d}_i < 0$, faça $\bar{d}_i = 0$, $i = 1, \ldots, m$.

Passo 8.2: Se $\bar{d}_i = 0 \ \ \forall \ \ i = 1, \ldots, m$, FIM. Caso contrário, faça $j = j + 1$ e volte ao Passo 2.

2.4 Modelo não-linear

Em 2001, Martínez [17] propôs uma suavização de funções do tipo da função $\delta(x_j)$ do Problema (2.8) para que um método de otimização não-linear contínua possa ser aplicado ao problema. Considere o problema:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{j=1}^{n} H_j(x_j) \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij}x_j = d_i \quad , \quad i = 1, \ldots, m \\ & x_j \geq 0 \quad , \qquad j = 1, \ldots, n, \end{aligned} \tag{2.10}$$

onde $x = (x_1, \ldots, x_n)^T$ e

$$H_j(t) = \begin{cases} c_j t + \delta_j, \text{ se } t > 0 \\ 0, \text{ caso contrário} \end{cases},$$

para $j = 1, \ldots, n$ e assumindo $\delta_j > 0$ e $c_j \geq 0$. Neste problema, o espaço solução é dado por $\Omega = \left\{ x \in I\!R^n \;\; | \;\; \sum_{j=1}^{n} a_{ij}x_j = d_i, i = 1, \ldots, m \text{ e } x \geq 0 \right\}$.

As aproximações para a função objetivo do problema (2.10) são dadas por:

$$H_{jk}(t) = \begin{cases} c_j t + \delta_j \dfrac{kt^2}{1 + kt^2}, \text{ se } t > 0 \\ 0, \text{ caso contrário.} \end{cases} \tag{2.11}$$

Um importante resultado teórico obtido em [17] e descrito em [20] é que se para todo $k = 0, 1, 2, \ldots$, x^k é solução global de (2.11) e $x^* \in \Omega$ é um ponto de acumulação de x^k, então x^* é solução de (2.10). Como o problema (2.11) tem função objetivo contínua, podemos aplicar métodos de otimização não-linear para sua resolução e encontrar uma solução aproximada para o problema (2.10).

2.4.1 Suavizando o termo não-linear

A função objetivo do Problema (2.9) é uma função descontínua, o que torna o problema difícil de ser resolvido. Para contornar este problema, seguimos a proposta de Martínez [17] e Salles Neto [20] e suavizamos a função objetivo. Além disso, relaxamos a condição de integralidade da solução do Problema (2.9). Chamaremos de P_k o seguinte problema:

$$\begin{aligned} \text{Minimizar} \quad & c_1 \sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \frac{k x_j^2}{1 + k x_j^2} + c_3 \sum_{j=1}^{n} t_j x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \qquad i = 1, \ldots, m \\ & x_j \geq 0, \qquad j = 1, \ldots, n. \end{aligned} \tag{2.12}$$

Note que o problema é dependente do valor de k. Para valores de k suficientemente grandes, a função objetivo do problema aproxima bem uma função do tipo $\delta(x_j)$.

A Figura 2.6 mostra os gráficos do termo não-linear da função objetivo de (2.12) variando $k = 0, 1, 10, 100, 1000, 10000$, no caso unidimensional. À medida que o valor de k aumenta nota-se a caracterização da função $\delta(x_j)$, isto é, para valores muito próximos de zero, a função vale 0 e caso contrário assume valor 1.

2.4.2 Globalização

Ao se resolver o Problema (2.12), é preciso encontrar o ótimo global. Porém, os algoritmos para resolução de problemas de otimização garantem em geral a obtenção de um ótimo local, que depende da aproximação inicial escolhida. Existem várias heurísticas em otimização global, por exemplo *Simulated Annealing* [15], algoritmos genéticos [1], busca tabu [4, 11, 12]. A seguir descrevemos uma destas heurísticas, *Simulated Annealing*.

Simulated Annealing (SA)

O *Simulated Annealing* (SA) é uma heurística de otimização global baseada na mecânica estatística de recozimento de materiais, [15]. Na metalurgia, *annealing* ou recozimento é o processo em que uma peça de metal é submetida a uma temperatura inicial alta e então resfriada lentamente para que o produto final fique homogêneo. Esse resfriamento é mais

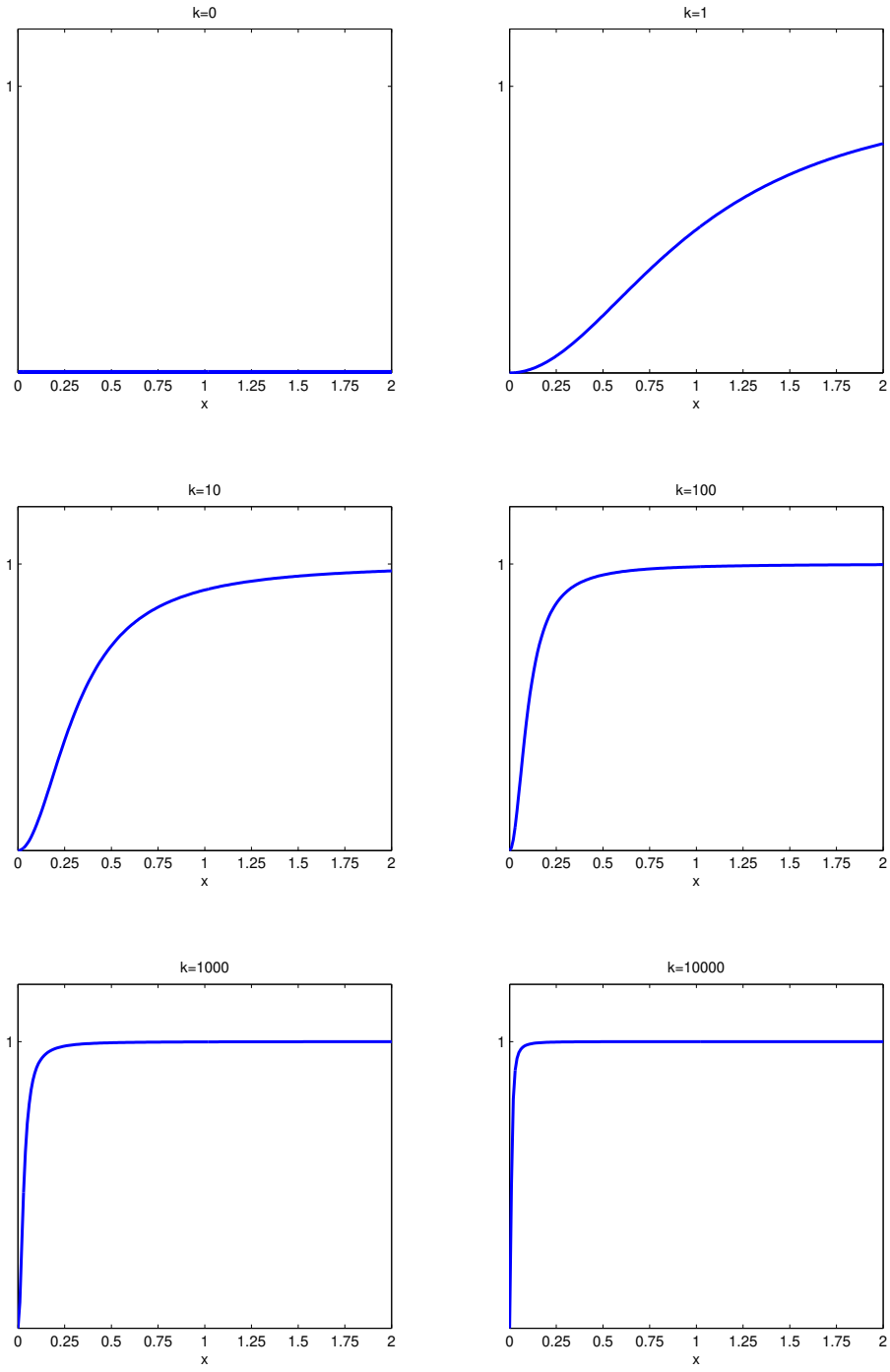


Figura 2.6: Comportamento de $\frac{kx^2}{1+kx^2}$, $k = 0, 1, 10, \ldots, 10000$.

acentuado inicialmente e ao longo do processo a temperatura decai mais lentamente.

Considere o seguinte problema:

$$\begin{aligned} \text{Minimizar} \quad & f(x) \\ \text{sujeito a} \quad & x \in \Omega, \end{aligned} \tag{2.13}$$

onde $\Omega = \{x \in I\!R^n \mid l \leq x \leq u\}$, em que l e u são, respectivamente, os vetores dos limitantes inferiores e superiores da variável x.

A heurística consiste em inicializar uma temperatura inicial e uma aproximação $x \in \Omega$. A partir de x são gerados pontos em sua vizinhança. O novo ponto é aceito ou rejeitado de acordo com o valor da função objetivo e de uma função de probabilidade que é dependente da temperatura. Nos estágios iniciais, a probabilidade de aceitação é maior, assim, mesmo que um novo ponto gerado não tenha um valor de função objetivo menor que os gerados anteriores, pode ser aceito. Dessa forma, tenta-se escapar dos minimizadores locais. O processo é repetido até que um certo equilíbrio é obtido e então a temperatura é reduzida, analogamente ao processo de recozimento em que a temperatura é diminuída quando se atinge um certo nível de energia dos átomos. Em seguida, o processo é executado novamente para esta nova temperatura. Efetuamos a implementação como descrito em [13] e [19].

Antes de inicializar o algoritmo é preciso estabelecer a temperatura inicial T_0. Para isso são geradas τ_0 transições aleatórias (pontos iniciais aleatórios) em que o valor da função objetivo aumenta (por exemplo $\tau_0 = 50$). Calcula-se a temperatura inicial pela seguinte expressão:

$$P_0 = exp\left(-\frac{\Delta f_0^+}{T_0}\right),$$

onde Δf_0^+ é a média da variação absoluta das τ_0 transições aleatórias, e P_0, a probabilidade inicial de aceitação é usualmente fixada em $0,5$.

A vizinhança $V_k(x)$ é o conjuntos dos pontos $y \in I\!R^n$ tais que $x_i - \lambda_i \leq y_i \leq x_i + \lambda_i$, para $i = 1, \ldots, n$. O tamanho λ desta vizinhança é atualizado a cada troca de temperatura.

Existem muitas formas de se atualizar a temperatura. Como em [13] e [19], utilizamos

Algoritmo 2 *Simulated Annealing.*

Passo 0: Dados x, n, N_1, N_2 e T_0.

Passo 1: Faça $k = 0$, $a = 0$ e $g = 0$.

Repita os Passos 2-4 até que um critério de parada seja satisfeito.

Passo 2: Enquanto $a < N_1 \times n$ e $g < N_2 \times n$:

Passo 2.1: Gere $y \in V_k(x)$ e faça $g = g + 1$.

Passo 2.2: Se $f(y) \leq f(x)$, faça $x = y$ e $a = a + 1$.

Caso contrário, gere w, um número aleatório em $[0, 1)$ e se $w < \exp\left(\frac{f(x)-f(y)}{T_k}\right)$ faça $x = y$ e $a = a + 1$.

Passo 3: Faça $k = k + 1$, $g = 0$ e $a = 0$.

Passo 4: Calcule T_k.

para o decréscimo da temperatura $T_{k+1} = \alpha T_k$, onde α é o máximo entre β (usualmente $\beta = 0.1$) e o mínimo entre um parâmetro (tipicamente 0.9) e a razão do menor valor da função objetivo pela soma de todos os valores de função objetivo, ambos referentes à temperatura corrente T_k.

Os critérios para troca da temperatura atual T_k para um problema de dimensão n são:

- quando a quantidade de pontos aceitos for maior que $N_1 \times n$ (tipicamente $N_1 = 12$);
- quando a quantidade de pontos gerados for maior que $N_2 \times n$ (tipicamente $N_2 = 100$).

Os critérios de parada do algoritmo utilizados são:

- se durante as últimas μ (por exemplo $\mu = 4$) trocas de temperatura não houver melhora no valor da função objetivo;
- se a temperatura atual atingiu a temperatura final estimada;
- se o tamanho de passo, entre o ponto gerado e o último ponto aceito, é menor que uma precisão baseada nos limitantes das variáveis;
- se o número máximo de avaliações de função é atingido;
- se o número máximo de trocas de temperatura é atingido.

O *Simulated Annealing* foi proposto para problemas irrestritos ou com restrições de caixa. No nosso caso, como o problema tem restrições, vamos incorporá-las na função objetivo através de uma penalização conforme [13]:

$$f_{\mathrm{sa}}(x) = c_1 \sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \frac{k x_j^2}{1 + k x_j^2} + c_3 \sum_{j=1}^{n} t_j x_j + \rho \sum_{i=1}^{m} \max\{0, d_i - \sum_{j=1}^{n} a_{ij} x_j\}$$

onde ρ é o parâmetro de penalização. Assim, se a demanda do item i é satisfeita, a penalização é zero. Caso contrário, quanto mais faltar para atender a demanda, maior será a penalização na função objetivo.

2.5 Método de Gilmore e Gomory para geração de colunas

Vimos na Seção 2.2 que uma das etapas na resolução do Problema (2.9), é a geração de um novo padrão, que equivale a gerar uma nova coluna para o problema.

Iniciamos a descrição do método de geração de colunas proposto por Gilmore e Gomory, [9, 10], resumindo a obtenção da solução primal de um problema de programação linear. A partir desta solução primal descrevemos como gerar uma coluna de modo que seja candidata a entrar como básica, melhorando a solução primal previamente obtida.

Neste resumo seguimos a notação usualmente adotada em programação linear, [2, 5].

Seja o problema primal de programação linear:

$$\begin{aligned} \text{Minimizar} \quad & f(x) = \gamma^T x \\ \text{sujeito a} \quad & Ax = d \\ & x \geq 0, \end{aligned} \tag{2.14}$$

onde $\gamma \in I\!R^n$ é o vetor custo dos objetos processados, $A \in I\!R^{m \times n}$ e posto$(A) = m$.

Considere uma partição nas colunas de A:

$$A = [B \;\; N],$$

tal que $B \in \mathbb{R}^{m\times m}$ é uma matriz formada por m colunas linearmente independentes de A e é denominada matriz básica.

Da mesma forma, particionamos o vetor das variáveis:

$$x = [x_B, x_N],$$

onde x_B é chamado de vetor das variáveis básicas e x_N é o vetor das variáveis não-básicas.

Podemos reescrever as restrições de igualdade da seguinte maneira:

$$Ax = d \Leftrightarrow Bx_B + Nx_N = d.$$

Como B é inversível, as variáveis básicas podem ser escritas em termos das não-básicas. Temos que:

$$Ax = d \Leftrightarrow x_B = B^{-1}d - B^{-1}Nx_N.$$

Assim, uma solução particular, chamada de solução básica, para o problema (2.14) é $x_B = B^{-1}d$ e $x_N = 0$. Se $x_B = B^{-1}d \geq 0$, então a solução básica é primal-factível e a partição é dita primal-factível.

Considere também a partição no vetor custo das variáveis:

$$\gamma = [\gamma_B, \gamma_N].$$

Escrevendo o dual do problema (2.14), temos:

$$\begin{aligned} \text{Maximizar} \quad & g(\pi) = d^T\pi \\ \text{sujeito a} \quad & A^T\pi \geq \gamma \\ & \pi \text{ irrestrito,} \end{aligned} \tag{2.15}$$

onde $\pi \in \mathbb{R}^m$ é o vetor das variáveis duais.

Da teoria da dualidade em programação linear, [5], temos que $\pi^T = \gamma_B^T B^{-1}$, e portanto,

a função objetivo do problema primal pode ser escrita como:

$$\begin{aligned} f(x) &= \gamma^T x \\ &= \gamma_B^T x_B + \gamma_N^T x_N \\ &= \gamma_B^T B^{-1} d + (\gamma_N^T - \pi^T N) x_N \\ &= \gamma_B^T B^{-1} d + \sum_{j \in I(N)} (\gamma_j - \pi^T a_j) x_j, \end{aligned} \tag{2.16}$$

onde $I(N)$ é o conjunto dos índices das variáveis não-básicas e a_j é a j-ésima coluna da matriz A.

Supondo que a solução ótima para o problema primal tenha sido obtida, o objetivo é gerar uma coluna, a_{n+1}, de modo que esta coluna seja candidata a entrar na base, isto é, de modo que a solução do problema deixe de ser ótima. Analisando (2.16), para que uma variável não-básica x_{n+1}, entre na base com o intuito de diminuir o valor da função objetivo, devemos ter:

$$\gamma_{n+1} - \pi^T a_{n+1} < 0. \tag{2.17}$$

Entretanto, na formulação do Problema (2.9), o custo γ_{n+1} que utilizamos depende da coluna a ser gerada, pois

$$t_{n+1} = \frac{W - w^T a_{n+1}}{W}.$$

Reescrevendo o custo relativo temos:

$$\begin{aligned} \gamma_{n+1} - \pi^T a_{n+1} &= c_1 + c_3 t_{n+1} - \pi^T a_{n+1} \\ &= c_1 + c_3 \left(\frac{W - w^T a_{n+1}}{W} \right) - \pi^T a_{n+1} \\ &= c_1 + c_3 - \left(\frac{c_3}{W} w + \pi \right)^T a_{n+1} \\ &= (c_1 + c_3) - \eta^T a_{n+1}, \end{aligned} \tag{2.18}$$

onde $\eta = \frac{c_3}{W} w + \pi \in I\!R^m$.

Seja o problema de programação linear dado por:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{j=1}^{n} \gamma_j x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i \quad , \quad i = 1, \ldots, m \\ & x_j \geq 0 \text{ e inteiro}, \quad j = 1, \ldots, n, \end{aligned} \tag{2.19}$$

onde $\gamma \in I\!R^m$ é o vetor custo dos objetos processados.

O método proposto por Gilmore e Gomory consiste em formular e resolver um Problema da Mochila, no qual a restrição corresponde à exigência de que o padrão tem que ser factível e, a função objetivo é construída de modo a favorecer que a Condição (2.18) seja satisfeita pelo padrão a_{n+1} que será gerado. Segue então o Problema da Mochila Ilimitada (PMI):

$$\begin{aligned} \text{Maximizar} \quad & Z = \sum_{i=1}^{m} \eta_i a_i \\ \text{sujeito a} \quad & \sum_{i=1}^{m} w_i a_i \leq W \\ & a_i \geq 0 \text{ e inteiro}, \quad i = 1, \ldots, m, \end{aligned} \tag{2.20}$$

onde $\eta = \frac{c_3}{W} w + \pi \in I\!R^m$ e $\pi \in I\!R^m$ é o vetor dos multiplicadores (variáveis duais) da solução do problema (2.19). Para que a coluna entre na base, devemos ter $(c_1 + c_3) - \eta^T a < 0$, ou seja, $Z = \eta^T a > c_1 + c_3$.

2.5.1 Multiplicadores de Lagrange

No método de geração de colunas de Gilmore e Gomory, vimos que a formulação do problema da mochila requer as variáveis duais da solução do problema linear original. Ocorre que o problema P_k (2.12) tem função objetivo não-linear e a dificuldade é como generalizar a proposta de Gilmore e Gomory para o caso não-linear.

A fim de utilizar o método de geração de colunas de Gilmore e Gomory para o problema não-linear, Salles Neto [20] apresentou uma formulação linear para P_k, chamado de

Problema Linear Auxiliar 1 (PLA1):

$$\text{Minimizar} \quad \sum_{j=1}^{n} \gamma_j x_j - \sum_{j=1}^{p} y_j \tag{2.21}$$

$$\text{sujeito a} \quad \sum_{j=1}^{n} a_{ij} x_j \geq d_i, \quad i = 1, \ldots, m \tag{2.22}$$

$$\sum_{j=1}^{n} y_j = p \tag{2.23}$$

$$L y_j - x_j \geq 0, \quad j = 1, \ldots, n \tag{2.24}$$

$$0 \leq y_j \leq 1, \quad j = 1, \ldots, n \tag{2.25}$$

$$x_j \geq 0, \quad j = 1, \ldots, n, \tag{2.26}$$

onde p é o número de padrões distintos na solução de P_k, $\gamma_j = c_1 + c_3 t_j$ é o custo da variável x_j, $j = 1, \ldots, n$, e L é um número real grande.

A diferença neste trabalho, em relação ao modelo PLA1 em [20] é o custo γ_j, que em [20] é $\gamma_j = c_1 = 1$, $j = 1, \ldots, n$.

Nesta formulação, o vetor $y \in I\!R^n$ representa o *setup*. A Equação (2.24) determina que se $y_j = 0$, então o padrão j não entra na solução, isto é, a variável correspondente x_j é nula. Além disso, a garantia de que haverá apenas p padrões na solução é dada pela Equação (2.23), e juntamente com a Equação (2.25), determina que y_j seja 0 ou 1, para $j = 1, \ldots, n$ e então, $\sum_{j=1}^{n} y_j = \sum_{j=1}^{p} y_j = p$. É possível eliminar o vetor y e $n-p$ componentes do vetor x. Logo, podemos reescrever PLA1 de uma forma mais simplificada, reordenando as variáveis para que as p primeiras correspondam àquelas que são estritamente positivas, chamado de Problema Linear Auxiliar 2 (PLA2):

$$\begin{aligned} \text{Minimizar} \quad & \sum_{j=1}^{p} \gamma_j x_j \\ \text{sujeito a} \quad & \sum_{j=1}^{p} a_{ij} x_j \geq d_i, \quad i = 1, \ldots, m \\ & x_j \geq 0, \quad j = 1, \ldots, p. \end{aligned} \tag{2.27}$$

Mostra-se que se $x^* \in I\!R^n$ é solução do problema P_k, com as p primeiras componentes

não nulas e $\overline{x} \in I\!R^p$ é solução de PLA2, então

$$\sum_{j=1}^{n} \gamma_j x_j^* = \sum_{j=1}^{p} \gamma_j \overline{x}_j.$$

Porém, observamos que quando ocorre o caso de infinitas soluções ótimas para o problema primal, nem sempre teremos $\overline{x}_j = x_j^*$, $j = 1, \ldots, p$. O que se pode afirmar é que a solução do problema P_k é, considerando-se apenas as p componentes não nulas, solução de PLA2. Com isso, dado x solução de P_k, que é também solução de PLA2 para as p componentes não nulas, faremos a seguir uma comparação dos multiplicadores de Lagrange (variáveis duais), λ_j e μ_j, de ambos os problemas.

Escrevendo as condições KKT do problema P_k (2.12) temos:

- $\nabla f(x)_j + \sum_{i=1}^{m} a_{ij}\lambda_i + \mu_j = 0 \ , \quad j = 1, \ldots, n$
- $\lambda_i \leq 0 \ , \quad i = 1, \ldots, m.$
- $\lambda_i \left(\sum_{j=1}^{n} a_{ij} x_j - d_i \right) = 0 \ , \quad i = 1, \ldots, m$
- $\mu_j x_j = 0 \ , \quad j = 1, \ldots, n$

Suponha que as p primeiras componentes da solução x de P_k são não nulas. Assim, $x_j = 0$, $j = p+1, \ldots, n$ e $\mu_j = 0$, $j = 1, \ldots, p$. Reescrevendo as condições KKT, temos:

- $c_1 + c_2 \dfrac{2kx_j}{(1+kx_j^2)^2} + c_3 t_j + \sum_{i=1}^{m} a_{ij}\lambda_i = 0 \ , \quad j = 1, ..., p$

  $c_1 + c_3 t_j + \sum_{i=1}^{m} a_{ij}\lambda_i + \mu_j = 0 \ , \quad j = p+1, \ldots, n$
- $\lambda_i \leq 0 \ , \quad i = 1, \ldots, m$
- $\lambda_i \left(\sum_{j=1}^{p} a_{ij} x_j - d_i \right) = 0 \ , \quad i = 1, \ldots, m$

Para PLA2, as componentes da solução são estritamente maiores que zero, logo $\mu = 0$. Podemos escrever as condições KKT, já simplificadas:

- $c_1 + c_3 t_j + \sum_{i=1}^{m} a_{ij}\lambda_i = 0 \ , \quad j = 1, \ldots, p$
- $\lambda_i \leq 0 \ , \quad i = 1, \ldots, m$

- $\lambda_i \left(\sum_{j=1}^{p} a_{ij} x_j - d_i \right) = 0 \ , \quad i = 1, \ldots, m$

Nota-se que as condições KKT de P_k e de PLA2 são parecidas. A diferença está no termo do gradiente referente à parte não-linear da função objetivo de P_k.

O gradiente da função objetivo de P_k é dado por

$$\nabla f(x) = c_1 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} \frac{2kx_1}{(1+kx_1^2)^2} \\ \vdots \\ \frac{2kx_n}{(1+kx_n^2)^2} \end{bmatrix} + c_3 \begin{bmatrix} t_1 \\ \vdots \\ t_n \end{bmatrix}$$

A Figura 2.7 mostra o comportamento de $\frac{2kx}{(1+kx^2)^2}$. Para k suficientemente grande, observamos que a função assume valores positivos somente para valores de x muito próximos de zero. Para $x \geq 1$, o valor deste termo está próximo de zero. Para o Problema (2.9), procuramos valores de x inteiros, assim gostaríamos de ter valores de x ou zero ou maiores do que 1. Dessa forma, os multiplicadores dos dois problemas têm valores próximos para uma mesma solução. Com este argumento, utilizaremos os multiplicadores obtidos no problema não-linear P_k para montar o problema da mochila na geração de uma coluna nova, conforme será descrito nas próximas subseções, o que difere o algoritmo daquele proposto em [20].

2.5.2 Resolução do Problema da Mochila

Nesta subseção descrevemos três maneiras de resolver o problema da mochila. A primeira delas é o Método Branch & Bound, [5]. A segunda, a qual fazemos uma breve descrição, é a Heurística de Martello e Toth [16]. Por fim, apresentamos nossa proposta, Problema da mochila modificado.

Método Branch & Bound (B&B)

Um método para resolução do problema da mochila muito utilizado é o Método Branch & Bound, B&B. A seguir, resumimos este método seguindo o texto de Chvátal, [5].

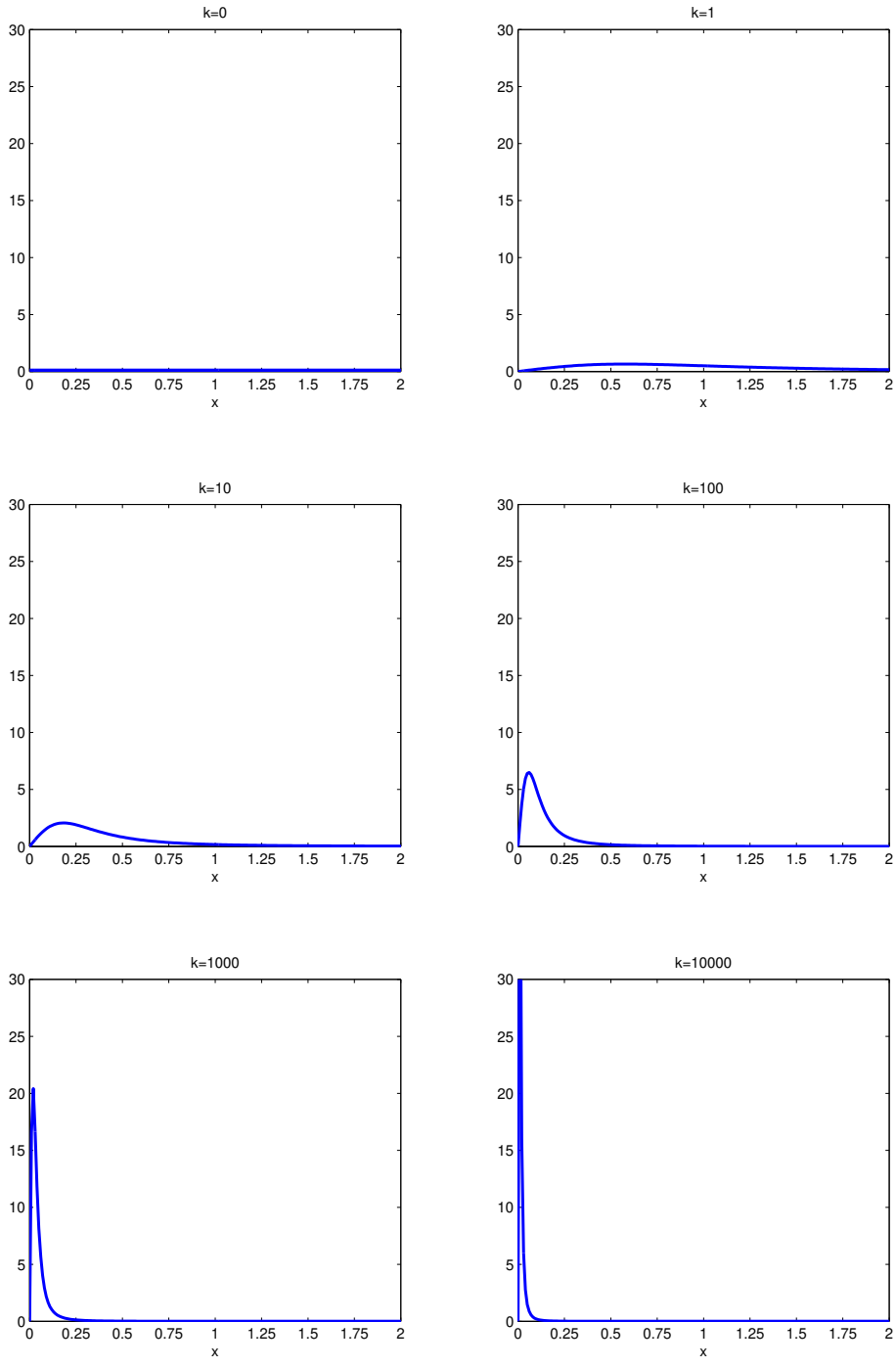


Figura 2.7: Comportamento de $\dfrac{2kx}{(1+kx^2)^2}$, $k = 0, 1, 10, \ldots, 10000$.

Considere o problema da mochila ilimitada (2.20). Cada item a_i que entra na mochila tem peso w_i e utilidade η_i. Define-se a razão η_i/w_i como a eficiência da variável a_i.

Sem perda de generalidade, suponha que:

$$\frac{\eta_1}{w_1} \geq \frac{\eta_2}{w_2} \geq \ldots \geq \frac{\eta_m}{w_m}. \tag{2.28}$$

Note que toda solução ótima satisfaz $W - \sum_{i=1}^{m} w_i a_i < a_m$. Se a solução não satisfaz essa inequação, implica que ainda é possível aumentar em pelo menos uma unidade a variável a_m, melhorando o valor da função objetivo.

Seja a melhor solução corrente $a_1^*, a_2^*, \ldots, a_m^*$. Definimos S como

$$S = \sum_{i=1}^{m} \eta_i a_i^*.$$

Suponha que ao fazer um movimento de volta ("*backtracking*") de algum $a_1, a_2, \ldots, a_m$, obtemos o maior k tal que $k \leq m-1$ e $a_k > 0$, e fazemos:

$$\begin{aligned} \overline{a}_i &= a_i \ , \quad i = 1, 2, \ldots, k-1, \\ \overline{a}_k &= a_k - 1. \end{aligned}$$

Examinamos as várias escolhas de $\overline{a}_{k+1}, \overline{a}_{k+2}, \ldots, \overline{a}_m$ que levam para as diferentes soluções $\overline{a}_1, \overline{a}_1, \ldots, \overline{a}_m$.

Por (2.28) sabemos que cada uma das variáveis $a_{k+1}, a_{k+2}, \ldots, a_m$ tem eficiência pelo menos η_{k+1}/w_{k+1}, e portanto podemos escrever:

$$\sum_{i=k+1}^{m} \eta_i \overline{a}_i \leq \frac{\eta_{k+1}}{w_{k+1}} \sum_{i=k+1}^{m} w_i \overline{a}_i.$$

Assumindo que a solução $\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_m$ é viável, isto é, $\sum_{i=1}^{m} w_i \overline{a}_i \leq W$, temos:

$$\sum_{i=1}^{m} \eta_i \overline{a}_i \leq \sum_{i=1}^{k} \eta_i \overline{a}_i + \frac{\eta_{k+1}}{w_{k+1}} \left(W - \sum_{i=1}^{k} w_i \overline{a}_i \right). \tag{2.29}$$

Apenas se o lado direito de (2.29) for estritamente maior que S, há chance de uma

das soluções $\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_m$ ser melhor do que $a_1^*, a_2^*, \ldots, a_m^*$. Logo, nenhuma das soluções $\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_m$ pode ser melhor que a solução corrente se a seguinte condição for satisfeita:

$$\sum_{i=1}^{k} \eta_i \overline{a}_i + \frac{\eta_{k+1}}{w_{k+1}} \left(W - \sum_{i=1}^{k} w_i \overline{a}_i \right) \leq S. \tag{2.30}$$

O Algoritmo 3 descreve o Método Branch & Bound para resolução do problema da mochila ilimitada.

Algoritmo 3 Branch & Bound para resolução do Problema da Mochila Ilimitada.

Passo 1: [Inicialize] Faça $S = 0$, $k = 0$.

Passo 2: [Encontre a extensão mais promissora do ramo atual]

Para $j = k+1, k+2, \ldots, m$, faça $a_j = \left\lfloor \left(W - \sum_{i=1}^{j-1} w_i a_i \right) / w_j \right\rfloor$.

Faça $k = m$.

Passo 3:[Uma melhor solução pode ser obtida?]

[Não] Se $\sum_{i=1}^{m} \eta_i a_i \leq S$, ir para o Passo 4.

[Sim] Caso contrário, faça $S = \sum_{i=1}^{m} \eta_i a_i$ e $a_i^* = a_i$, $i = 1, \ldots, m$.

Passo 4:[*Backtracking* para o próximo ramo]

(a) Se $k = 1$, FIM; caso contrário, faça $k = k - 1$.

(b) Se $a_k = 0$, volte para (a); caso contrário, faça $a_k = a_k - 1$.

Passo 5:[Vale a pena explorar este ramo?]

[Não] Se $\sum_{i=1}^{k} \eta_i a_i + \frac{\eta_{k+1}}{w_{k+1}} \left(W - \sum_{i=1}^{k} w_i a_i \right) \leq S$, volte para o Passo 4.

[Sim] Caso contrário, volte para o Passo 2.

Heurística de Martello e Toth (MTB2)

A heurística proposta por Martello e Toth, [16], chamada de MTB2, é utilizada para resolução do Problema da Mochila Limitada (PML):

$$\begin{aligned} \text{Maximizar} \quad & Z = \sum_{i=1}^{m} \eta_i a_i \\ \text{sujeito a} \quad & \sum_{i=1}^{m} w_i a_i \leq W \\ & 0 \leq a_i \leq b_i \text{ e inteiro}, \;\; i = 1, \ldots, m, \end{aligned} \tag{2.31}$$

onde b_i inteiro é o limitante superior para a variável a_i, $i = 1, \ldots, m$.

Esta heurística primeiramente transforma o Problema (2.31) em um problema da mochila 0-1. Para a resolução do problema da mochila 0-1, utilizam o método Branch & Bound. Ver detalhes em [16].

Problema da mochila modificado

No procedimento de gerações de padrões iniciais de Haessler, descrito na Subseção 2.3.2, alguns critérios para aceitação de um padrão gerado são exigidos a fim de se obter padrões de forma a minimizar o desperdício e o *setup*. Baseados nesta idéia de, ao se construir um padrão, exigir alguns critérios ou características para o mesmo, propomos uma modificação na formulação do problema da mochila.

Nesta proposta, ao se formular o problema da mochila, queremos considerar não somente os custos das variáveis duais. Ao se considerar o *setup*, observamos que uma característica dos padrões que compõem a solução do problema é que não sejam esparsos, isto é, que tenham um número maior de diferentes itens. A explicação é que quanto mais itens diferentes são produzidos por padrão, menos padrões serão necessários para produzir todos os itens para atender a demanda. Portanto, para favorecer que o padrão produzido não seja esparso, adicionamos um termo na função objetivo de (2.20) de modo que cada item diferente que entre no padrão tenha um acréscimo de α na função objetivo do problema da mochila. Isto sugere a utilização da aproximação não-linear como foi feita para o problema de corte, pois segue a idéia de que se o item entra na mochila, então seu valor na função objetivo vale 1, caso contrário, vale zero. A diferença em relação ao problema

da mochila é que buscamos padrões que tenham maior número de diferentes itens. Além desta modificação na função objetivo, relaxamos a condição de integralidade da solução, resultando no seguinte problema:

$$\begin{aligned} \text{Maximizar} \quad & \sum_{i=1}^{m} \eta_i a_i + \alpha \sum_{i=1}^{m} \frac{k a_i^2}{1 + k a_i^2} \\ \text{sujeito a} \quad & \sum_{i=1}^{m} w_i a_i \leq W \\ & a_i \geq 0, \quad i = 1, \ldots, m, \end{aligned} \tag{2.32}$$

onde $\alpha \in I\!R$.

Chamamos de GCN, Geração de Coluna Não-Linear, quando utilizamos o modelo não-linear proposto com $\alpha > 0$, e GCL, Geração de Coluna Linear, para o problema na forma linear, isto é, $\alpha = 0$. A resolução do Problema (2.32) é realizada através de um algoritmo de programação não-linear.

Como é necessária uma solução inteira, devemos arredondar a solução. Neste trabalho, propomos a heurística de arredondamento descrita a seguir.

Heurística para arredondamento da solução do Problema da mochila modificado

Esta heurística de arredondamento tem como objetivo favorecer que o padrão seja o mais denso possível, reforçando a idéia da formulação do Problema (2.32).

Seja a solução do Problema (2.32). Ordenamos a solução tal que $a_1 \geq a_2 \geq \ldots \geq a_m$ e separamos as variáveis em três grupos:

- $G_{11} = \{a_i \text{ tal que } a_i \geq 1\}$;

- $G_{01} = \{a_i \text{ tal que } 0 < a_i < 1\}$;

- $G_{00} = \{a_i \text{ tal que } a_i = 0\}$.

Primeiramente, arredondamos as variáveis para o inteiro mais próximo. Se a capacidade da mochila for ultrapassada, isto é, $r = \sum_{i=1}^{m} w_i a_i > W$, então diminuimos em uma unidade, começando da componente com maior valor, dos itens pertencentes ao grupo G_{11}.

Se ainda é possível colocar mais itens na mochila, então começamos a colocar os itens do grupo G_{01}, que na solução de (2.32) são não-nulos.

2.6 Critério de parada

Dado um problema de corte, geramos um conjunto inicial de padrões, resolvemos o problema não-linear P_k e geramos um novo padrão. Cada padrão gerado é incorporado ao conjunto de padrões e o problema não-linear P_k é resolvido novamente. O critério de parada do processo está relacionado ao resultado obtido no procedimento de geração de padrões. No caso linear, se o padrão gerado for tal que seu custo relativo é negativo, então não entrará na base e portanto não há como melhorar a solução atual. Neste caso, a solução ótima foi obtida. Adotamos, no caso não-linear (P_k) o mesmo critério, destacando que a solução obtida é um ótimo local, pois não se pode afirmar que não há mais padrões promissores. Também não podemos garantir que um mesmo padrão não seja gerado. Portanto, além do critério do custo relativo, o processo é parado quando o padrão gerado for o mesmo que o gerado na iteração anterior.

Quando não há mais padrões a serem incorporados ao problema, devemos integralizar a solução que foi obtida nas variáveis reais. A seguir apresentamos algumas heurísticas de arredondamento para a solução.

2.7 Heurísticas de arredondamento

Ao final do processo resolução do problema de corte suavizado P_k e quando não há mais padrões a serem incorporados, obtemos a solução do problema de corte, porém nas variáveis contínuas. Entretanto, o problema requer uma solução inteira, e portanto, ao final do processo devemos arredondar a solução. A seguir estão apresentadas duas heurísticas de arredondamento propostas por Wascher e Gau, [24].

BRUSIM

A heurística BRUSIM consiste em arredondar para cima todas as variáveis, ou seja,

$$x_k = \lceil x_k \rceil, \quad k = 1, \ldots, n$$

BRURED

Outra heurística de arredondamento é a heurística BRURED. Primeiramente, arredonda-se todas as variáveis para cima, como em BRUSIM. Em seguida, para $k = 1, \ldots, n$, se $a_{ik}(x_k - 1) + \sum_{j=1, j\neq k}^{m} a_{ij}x_j \geq d_i$, $i = 1, \ldots, m$, então $x_k = x_k - 1$.

BRURED Modificada

O arredondamento da solução pode influenciar no *setup*, pois se uma variável têm valor entre 0 e 1, com o arredondamento pode assumir valor zero ou 1. Portanto, para diminuir o *setup* devemos aumentar o número de variáveis que assumem valor 0 com o arredondamento. Baseado neste conceito, propusemos uma modificação da heurística BRURED.

A modificação consiste em utilizar a heurística de arredondamento BRURED, porém ordenando as variáveis de modo que $x_1 \leq x_2 \leq \ldots \leq x_n$. Desta forma, ao diminuir em uma unidade as variáveis com valores menores, podemos tentar que elas se tornem nulas, diminuindo o *setup*.

2.8 Sobre o custo de *setup*

Ao invés de fixarmos o custo de *setup* c_2 como sendo 100 ou 300, conforme [20], podemos considerá-lo como um parâmetro dinâmico. Buscamos obter na função objetivo de P_k, o mesmo peso para o termo linear

$$c_1 \sum_{j=1}^{n} x_j + c_3 \sum_{j=1}^{n} t_j x_j = \sum_{j=1}^{n} (c_1 + c_3 t_j) x_j,$$

e o termo não-linear

$$c_2 \sum_{j=1}^{n} \frac{kx_j^2}{1 + kx_j^2}.$$

Inicialmente, temos obj_{SHP} o número de objetos processados e n_{SHP} o número de padrões gerados na solução inicial obtida por SHP. Considerando $c_1 = 1$ e $c_3 = 10$, para que os termos linear e não-linear tenham o mesmo peso, devemos ter $c_2 = obj_{SHP}/n_{SHP}$. Baseando-se nesta idéia e estimando que o número de objetos processados na solução final seja obj_{SHP}, propomos que

$$c_2 = \frac{obj_{SHP}}{n},$$

onde n é o número de padrões no problema corrente P_k.

Dessa forma, c_2 é um parâmetro dinâmico que a cada padrão gerado tem seu valor diminuído, favorecendo a minimização do número de objetos processados e o desperdício relativo.

Capítulo 3

Implementação e Testes numéricos

Neste capítulo detalhamos a implementação computacional dos métodos descritos no capítulo anterior e realizamos testes numéricos.

Chamamos de MNLPC a implementação do método com a formulação não-linear para o problema de corte, proposta deste trabalho. A seguir, temos um algoritmo básico de MNLPC.

Algoritmo 4 Algoritmo básico do Método MNLPC.

Passo 0: Dados : W, w, d, c_1, c_2, c_3.
Passo 1: Gere um conjunto de padrões iniciais.
Passo 2: Monte e obtenha a solução global de P_k, $k = 1, 10, 10^2, 10^3, 10^4$.
Passo 3: Utilize os multiplicadores da solução de P_k para formular o problema da mochila e gerar um novo padrão.
Passo 4: Se o padrão gerado é promissor ($Z \geq c_1 + c_3$), diferente do padrão gerado na iteração anterior e o número máximo de padrões gerados não for atingido, insira-o ao problema P_k e volte ao Passo 2. Caso contrário, ir para o Passo 5.
Passo 5: Obtenha uma solução inteira, empregando a heurística BRURED modificado.

No Passo 1 podem ser utilizados a solução inicial homogênea ou a heurística SHP. As técnicas de globalização utilizadas no Passo 2 foram a de chutes aleatórios e *Simulated Annealing* (SA). Para o Passo 3, as opções de resolução do problema da mochila são MTB2, B&B , GCN e GCL, descritos na Subseção 2.5.2.

O Método MNLPC, as heurísticas SHP e SA foram implementados na linguagem For-

tran. As demais implementações também são na mesma linguagem. Os experimentos numéricos foram executados em um computador biprocessado com processadores AMD Opteron 242, 1.5 GB RAM.

A implementação da heurística SHP foi conforme descrita na Subseção 2.3.2. Os parâmetros utilizados foram os citados na mesma.

Ao montar o problema P_k, utilizamos também um limitante superior para as variáveis, exigência do pacote de otimização usada. Definimos este limitante, igual para todas as variáveis, como

$$u = \max \{d_i, i = 1, \dots m\}.$$

A idéia de se colocar este limitante é que mesmo que o padrão seja formado por apenas um item, o número de vezes que este será utilizado não exceda a sua demanda. Tomando o maior deles, nenhum padrão será utilizado mais vezes que este máximo.

Definimos o número máximo de colunas geradas como sendo o triplo do número de colunas geradas na solução inicial.

Como limitantes para o problema da mochila utilizamos a proposta de Salles Neto, [20]. Seja p o número de padrões distintos existentes na solução corrente, processando NO objetos. Suponha que queremos diminuir em 20% o *setup*, então a estimativa média do número de objetos que o novo padrão a ser gerado deve processar é $MOP = NO/0.8p$. Dessa forma, supondo que o número de objetos processados com o novo padrão é MOP, para que não se exceda a demanda nem a largura da bobina, devemos ter para cada peça i o seguinte limitante para o número de itens de tamanho w_i processados com este padrão:

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

Para o algoritmo MTB2, são exigidos alguns critérios sobre os dados de entrada, por exemplo, serem inteiros e estritamente maiores que zero. Os multiplicadores oriundos da solução de P_k podem estar no intervalo (0,1), e se anularão ao tomarmos sua parte inteira. Para evitar que se tenha multiplicadores com valor zero não satisfazendo os critérios para que o algoritmo MTB2 seja executado, utilizamos $\eta_i = \lfloor \eta_i \rfloor + 1$. No caso do limitante ser

zero, $b_i = 0$, trocamos por $b_i = 1$.

Para o algoritmo GCN, resolvemos o Problema (2.32) utilizando $\alpha = 100$ e $k = 10^4$. Um dos critérios de parada do algoritmo é se o padrão não é promissor ($Z > c_1 + c_3$). Mesmo no caso de GCN, que possui função objetivo com um termo não-linear positivo, se $Z \leq c_1 + c_3$ o padrão é considerado promissor.

Para a resolução do problema com restrições (P_k e Problema (2.32)), utilizamos o pacote de otimização MINOS, [18]. Este pacote é utilizado para resolução de problemas de programação linear e problemas com função objetivo não-linear e restrições lineares.

Ao se utilizar a estratégia dos chutes aleatórios para resolver o problema P_k, para cada k resolvemos o problema com o MINOS para 20 chutes aleatórios dentro da caixa, ou seja, entre 0 e u. A solução do problema para cada valor de k será um dos chutes iniciais para o problema seguinte com $k = 10k$.

No *Simulated Annealing* (SA), os pontos aleatórios também são dados dentro da caixa. Definimos o número máximo de trocas de temperatura como sendo $\mu = 10$. Iniciamos com $k = 1$, e ao final do processo na temperatura corrente, utilizamos a solução para ser uma aproximação inicial para a resolução de P_k através do MINOS. Atualizamos então k ($k = 10k$), e iniciamos o processo para a nova temperatura. Terminado o processo, a solução será um dos chutes iniciais para o MINOS, além da solução encontrada para o problema anterior P_{k-1}. O valor de k aumenta até 10^4, ficando fixo para valores acima deste, mesmo que a temperatura no processo SA diminua.

Note que o processo SA e a resolução do problema não-linear com o MINOS são independentes. A solução obtida no SA a cada final do processo da temperatura corrente é usada como aproximação inicial para o MINOS. O processo de SA continua sem interferência da resolução de P_k pelo MINOS.

Utilizamos duas formas distintas para o SA: irrestrito e restrito. Na forma irrestrita, despreza-se a restrição de demanda, ou seja, trata-se o problema apenas com restrições de caixa. Já na forma restrita, incorporamos a restrição de demanda na função objetivo, como descrito na Subseção 2.4.2. Definimos o parâmetro de penalização $\rho = 100$.

Nas tabelas deste capítulo, *Setup* e Objetos indicam, respectivamente, o número de padrões distintos na solução e o número de objetos processados. A percentagem do des-

perdício é indicada por DESP., que é obtida considerando a perda total relativamente ao número de objetos processados:

$$\mathsf{DESP.} = 100 \times \frac{\sum_{j=1}^{n} t_j x_j}{\sum_{j=1}^{n} x_j} \quad ,$$

lembrando que $t_j = \dfrac{W - \sum_{i=1}^{m} a_{ij} w_i}{W}$ é a perda relativa que ocorre quando usamos o padrão j.

A percentagem do excesso de produção, relativamente ao número de objetos processados é dada por:

$$\mathsf{EXC.} = 100 \times \frac{\sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j - d_i \right) w_i}{W \sum_{j=1}^{n} x_j}.$$

Para a construção dos gráficos dos resultados dos testes numéricos, a variação de uma variável de comparação C de um método A em relação ao método B, é dado pela expressão $(C_A - C_B)/C_B$. Caso o problema vise minimizar a variável C, então se o resultado da expressão é negativo significa que o método A é melhor, se é positivo o método B é melhor e se é zero, ambos têm o mesmo desempenho.

3.1 Problemas gerados pelo CUTGEN1

Para os experimentos numéricos utilizamos o gerador de problemas CUTGEN1, de Gau e Wascher [8]. Os parâmetros para gerar um problema são os seguintes:

- m: dimensão do problema, que corresponde ao número de itens;
- W: largura do objeto (rolo mestre);
- v_1: limite inferior para o tamanho relativo dos itens em relação a W, ou seja, $w_i \geq v_1 W$, $i = 1, \ldots, m$;

Classe	v_1	v_2	m	$\overline{d}$
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 3.1: Classes geradas pelo CUTGEN1.

- v_2: limite superior para o tamanho relativo dos itens em relação a W, ou seja, $w_i \leq v_2 W$, $i = 1, \ldots, m$;
- $\overline{d}$: demanda média por item.

Em relação aos limitantes, devemos ter $0 < v_1 < v_2 \leq 1$.

Com esses parâmetros especificados $(m, W, v_1, v_2, \overline{d})$, são gerados problemas de uma mesma classe. Os tamanhos dos itens estão distribuídos uniformemente no intervalo $[v_1 W, v_2 W]$. O menor e maior itens têm tamanhos, respectivamente, $v_1 W$ e $v_2 W$. A demanda total D é dada por $D = m\overline{d}$ e é distribuída entre os diferentes itens.

Foram geradas 18 classes caracterizadas conforme mostra a Tabela 3.1. As classes de 1 a 6 são os problemas com itens pequenos, de 7 a 12 são os problemas com itens de larguras variadas e de 13 a 18 são os problemas com itens grandes.

Para cada classe foram geradas 100 problemas. As médias dos resultados para os problemas de CUTGEN1 utilizando o SHP estão mostradas na Tabela 3.2. O tempo zero

para a primeira classe indica que o tempo de execução foi menor que a precisão do relógio da máquina.

Classe	*Setup*	Objetos	Desp.	Tempo(s)
1	5.65	11.58	5.26	0.00000
2	7.69	110.88	1.05	0.00004
3	6.84	22.19	2.82	0.00004
4	9.54	217.20	0.83	0.00008
5	9.97	43.22	1.70	0.00016
6	14.86	427.71	0.81	0.00080
Média	9.09	138.80	2.08	0.00019
7	10.36	51.82	16.42	0.00008
8	11.52	517.86	16.44	0.00076
9	19.92	97.57	13.37	0.00040
10	21.52	983.04	14.20	0.00404
11	37.38	183.82	10.29	0.00228
12	41.13	1856.32	11.37	0.02352
Média	23.64	615.07	13.68	0.00518
13	10.14	64.89	19.43	0.00012
14	10.71	650.84	19.70	0.00072
15	19.13	123.57	16.77	0.00044
16	20.63	1238.83	17.12	0.00380
17	36.54	234.13	13.57	0.00244
18	39.90	2366.30	14.60	0.02408
Média	22.84	779.76	16.86	0.00527

Tabela 3.2: Médias de SHP.

3.2 Análise dos modelos para geração de padrões

Resolvemos os problemas de CUTGEN1 com o Método MNLPC, fixando-se $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$, para os diferentes esquemas de geração de coluna: MTB2, B&B, GCN e GCL. Observamos que c_3 foi fixado com 10, ao invés de custo 1, como o custo c_1, porque $t_j \in [0, 1)$ indica a percentagem do desperdício, ao multiplicar por 10 teremos o custo associado ao desperdício no intervalo $[0, 10)$. A estratégia de globalização utilizada foi a dos chutes aleatórios. As médias dos resultados dos testes estão mostradas nas Tabelas

3.3 e 3.4.

A Figura 3.1 mostra a variação de MTB2 em relação a B&B em termos de *setup*, objetos, desperdício e excesso de produção. Observamos que MTB2 obteve melhores resultados em relação ao *setup* para quase todas as classes, com exceção das classes 13, 17 e 18. Nestas, porém, a diferença a favor de B&B foi mínima. Em relação ao número de objetos, para as classes com itens pequenos, B&B teve melhor desempenho, exceto para a Classe 1. Nota-se que o desperdício foi menor para MTB2, entretanto o excesso de produção foi, em geral, maior, sendo que diferenças significativas ocorreram nas classes com itens pequenos.

A variação de GCN em relação a GCL é mostrada na Figura 3.2. Em relação ao *setup* e ao número de objetos ambos obtiveram desempenho semelhante. Para as classes com itens grandes, GCN foi superior a GCL em relação ao excesso de produção, porém inferior em relação ao desperdício.

A Figura 3.3 mostra a variação de GCN em relação a MTB2. Observamos que MTB2 foi superior em relação ao *setup* em todas as classes. Em relação ao número de objetos processados, GCN foi superior para quase todos os problemas das classes com itens pequenos. GCN foi superior em relação ao excesso, enquanto que em relação ao desperdício foi inferior.

A Tabela 3.5 mostra os tempos de execução, em segundos, de cada um dos métodos, fixados $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$. O Método MTB2 levou, para resolver os problemas com itens pequenos (classe 1 a 6), cerca de duas a três vezes mais tempo que os demais métodos. Para as outras classes de problemas, os métodos tiveram comportamento semelhante quanto ao tempo de execução. Na média geral, GCN e GCL apresentam tempo menor.

Os desperdícios obtidos, mostrados na Tabela 3.4 estão de acordo, em geral menor, com a média dos resultados obtidos pelo método ILS-APG, de Umetani, Yagiura e Ibaraki [21], para o mesmo conjunto de problemas. Para a maioria das classes, principalmente para as classes com itens maiores, a percentagem de desperdício de ILS-APG foi maior que 10%.

	MTB2		B&B		GCN		GCL	
Classe	*Setup*	Objetos	*Setup*	Objetos	*Setup*	Objetos	*Setup*	Objetos
1	2.79	17.45	3.36	18.80	3.33	19.03	3.36	18.96
2	3.33	139.62	3.50	137.05	3.48	137.85	3.49	138.92
3	4.36	31.74	4.89	29.02	4.90	29.42	4.92	28.94
4	4.80	258.90	4.94	254.10	4.91	255.84	4.92	254.69
5	7.05	60.65	7.61	55.39	7.62	56.13	7.63	54.55
6	8.06	495.14	8.38	485.52	8.46	479.59	8.44	480.48
Média	5.07	167.25	5.45	163.31	5.45	162.98	5.46	162.76
7	5.39	56.35	5.64	55.81	5.69	55.78	5.68	56.20
8	5.81	539.48	5.90	539.35	5.99	540.66	6.02	539.31
9	10.41	104.98	10.84	105.19	10.91	105.95	10.99	104.88
10	11.21	1012.42	11.34	1013.51	11.52	1018.58	11.57	1015.43
11	19.99	198.87	20.99	197.02	21.08	199.79	21.28	197.18
12	21.82	1919.02	22.13	1914.49	22.44	1919.35	22.48	1914.39
Média	12.44	638.52	12.81	637.56	12.94	640.02	13.00	637.90
13	6.68	67.61	6.64	68.04	6.86	67.35	6.74	68.01
14	6.84	668.89	6.87	668.33	7.02	666.71	6.99	667.24
15	12.50	128.70	12.63	128.58	12.92	128.07	12.81	128.65
16	13.08	1264.58	13.14	1266.19	13.46	1264.82	13.31	1267.37
17	23.75	242.98	23.61	243.88	24.32	242.68	23.95	243.93
18	24.81	2411.32	24.70	2410.06	25.61	2413.56	25.24	2424.84
Média	14.61	797.35	14.60	797.51	15.03	797.20	14.84	800.01

Tabela 3.3: Médias para $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

Classe	MTB2		B&B		GCN		GCL	
	Desp.	Exc.	Desp.	Exc.	Desp.	Exc.	Desp.	Exc.
1	2.56	32.43	2.11	34.77	1.89	35.62	2.05	35.39
2	0.66	18.78	0.60	17.35	0.59	17.34	0.60	17.89
3	1.16	27.53	1.22	22.15	1.19	22.96	1.17	21.86
4	0.60	14.95	0.62	13.46	0.62	14.02	0.62	13.67
5	0.77	25.91	0.89	19.82	0.80	20.78	0.83	18.93
6	0.67	13.22	0.68	11.65	0.68	10.66	0.68	10.71
Média	1.07	22.14	1.02	19.87	0.96	20.23	0.99	19.74
7	4.89	19.10	4.94	18.33	5.33	17.84	5.09	18.67
8	4.35	16.08	3.91	16.38	4.13	16.37	3.84	16.47
9	3.93	16.26	4.38	15.95	4.91	15.88	4.62	15.40
10	3.46	13.65	3.70	13.44	3.93	13.70	4.09	13.27
11	3.44	13.98	3.99	12.58	4.14	13.52	3.95	12.70
12	3.11	11.28	3.44	10.80	3.64	10.77	3.74	10.43
Média	3.86	15.06	4.06	14.58	4.35	14.68	4.22	14.49
13	8.84	14.21	9.17	14.35	10.09	12.70	9.69	13.76
14	8.63	13.52	8.41	13.59	9.64	12.20	9.01	12.90
15	6.85	13.55	7.12	13.14	8.55	11.41	7.61	12.70
16	6.51	12.51	6.17	12.90	8.10	10.94	6.94	12.24
17	5.81	11.09	6.05	11.12	6.94	9.82	6.21	10.98
18	5.71	10.59	5.59	10.68	6.93	9.41	6.11	10.61
Média	7.06	12.58	7.09	12.63	8.38	11.08	7.60	12.20

Tabela 3.4: Médias das perdas para $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

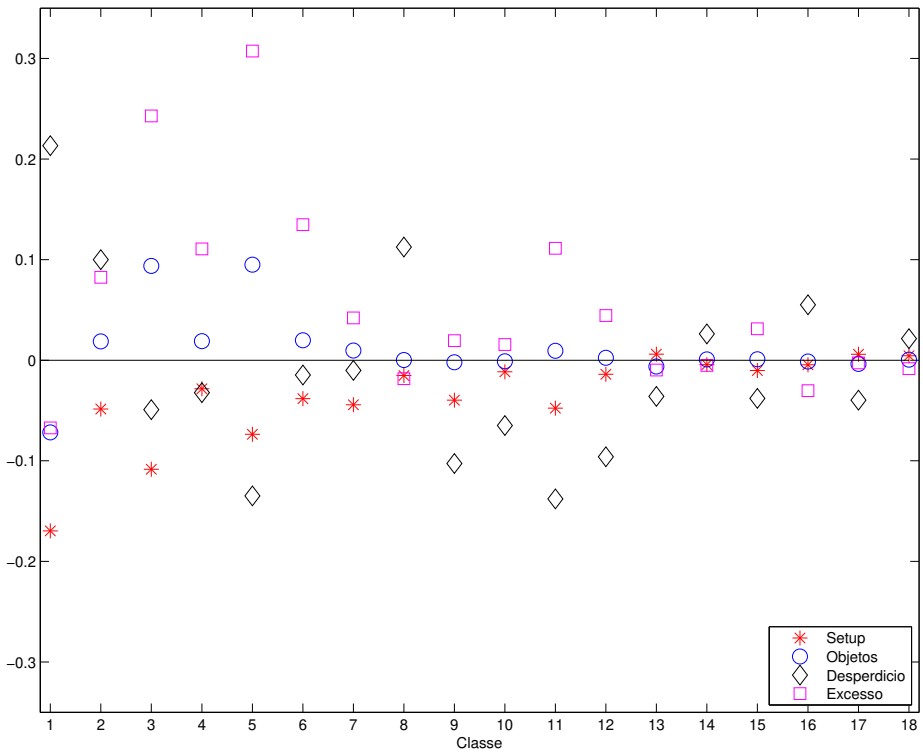


Figura 3.1: MTB2 em relação a B&B.

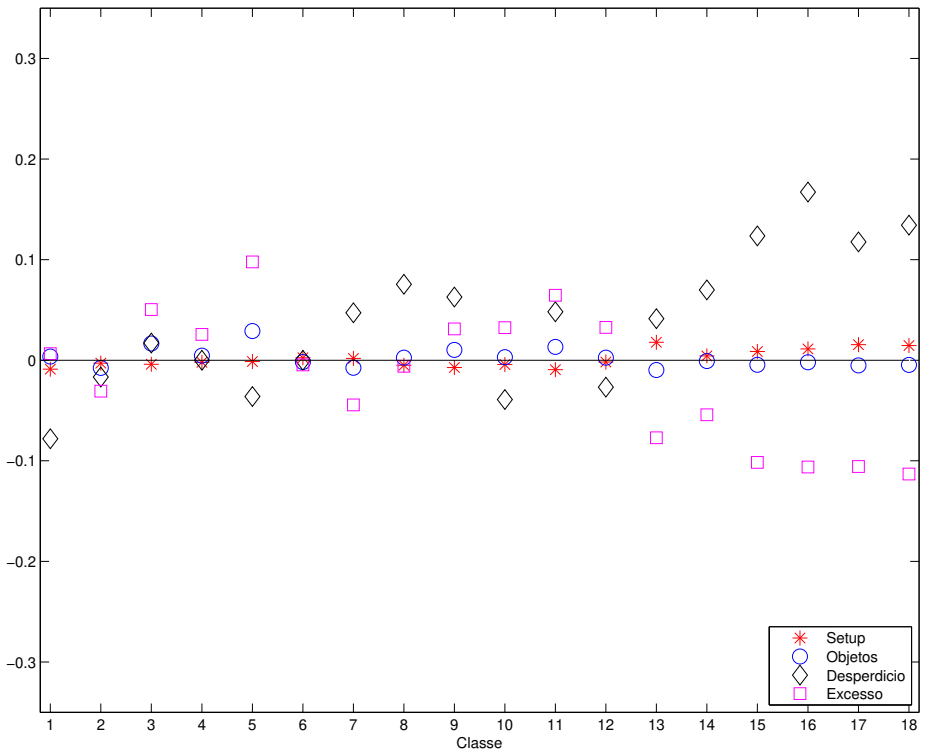


Figura 3.2: GCN em relação a GCL.

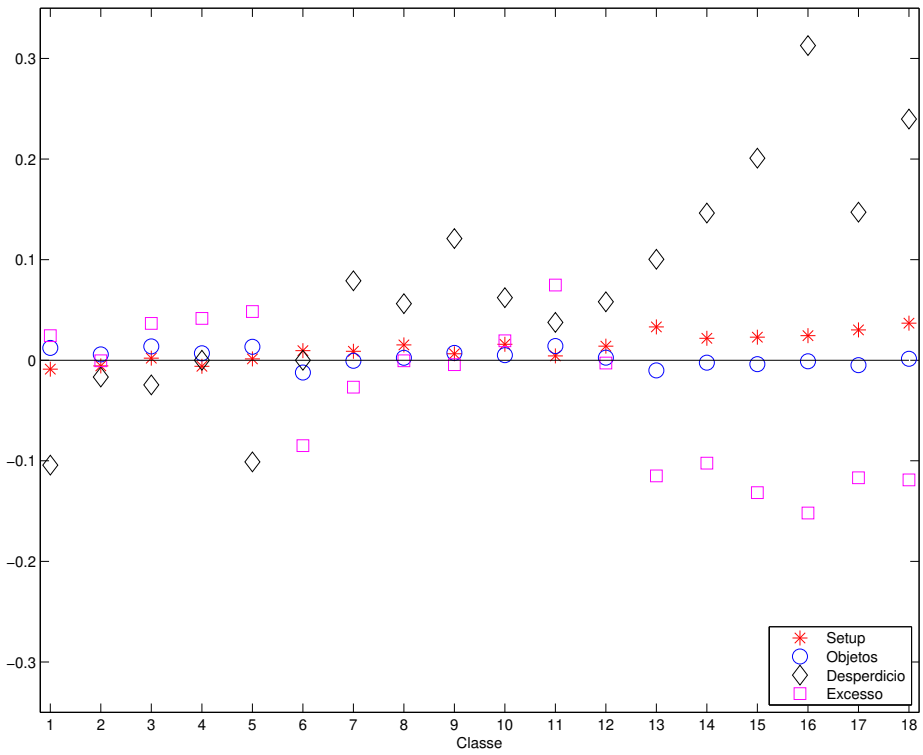


Figura 3.3: GCN em relação a MTB2.

Classe	MTB2 Tempo(s)	B&B Tempo(s)	GCN Tempo(s)	GCL Tempo(s)
1	0.24378	0.06348	0.07152	0.05912
2	0.29210	0.06736	0.07392	0.06692
3	0.37398	0.14221	0.15381	0.10049
4	0.40779	0.11353	0.10845	0.10757
5	0.77981	0.24630	0.31606	0.22373
6	0.69612	0.21445	0.27870	0.19129
Média	0.46560	0.14122	0.19708	0.12485
7	0.11785	0.10709	0.09909	0.09989
8	0.12773	0.12281	0.11997	0.11909
9	0.23825	0.19401	0.19073	0.18649
10	0.26614	0.23961	0.22829	0.21753
11	0.52071	0.32406	0.35698	0.31654
12	0.63676	0.52163	0.50407	0.48283
Média	0.31791	0.25154	0.24986	0.23706
13	0.11457	0.11209	0.09869	0.09953
14	0.11732	0.11881	0.10525	0.11173
15	0.26414	0.26930	0.21097	0.22073
16	0.30074	0.31894	0.23878	0.27234
17	0.66848	0.59028	0.50711	0.51191
18	0.78265	0.77813	0.65164	0.69736
Média	0.37465	0.36459	0.30207	0.31893

Tabela 3.5: Médias do tempo de execução, em segundos, para $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

3.2.1 Classe 8

Nesta subseção escolhemos um problema da Classe 8 para fazer um análise mais detalhada dos padrões gerados pelos 5 diferentes esquemas: SHP e o Algoritmo MNLPC com MTB2, B&B, GCN e GCL. Fixamos $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$. Esclarecemos que a solução obtida por SHP foi escolhida como solução inicial para todas as versões do Algoritmo MNLPC.

O problema escolhido tem 9 diferentes itens ($m = 9$), os tamanhos e as demandas dos itens são, respectivamente, $w = (794\ ,\ 761\ ,\ 651\ ,\ 466\ ,\ 376\ ,\ 304\ ,\ 281\ ,\ 149\ ,\ 91)^T$ e $d = (220\ ,\ 22\ ,\ 237\ ,\ 8\ ,\ 95\ ,\ 53\ ,\ 114\ ,\ 187\ ,\ 64)^T$.

Com a finalidade de analisar a eficiência dos padrões gerados em cada um dos esquemas,

resolvemos o problema P_k para o conjunto de todos os padrões gerados. Foram utilizados como chutes iniciais as soluções obtidas em cada um dos métodos e por mais 200 chutes iniciais aleatórios dentro da caixa, conforme descrito no início deste capítulo.

A Tabela 3.6 mostra os resultados obtidos em cada um dos métodos e a solução para o problema com todas os padrões gerados. Observamos que SHP não gera excesso de produção, porém, além do *setup*, o seu desperdício também é muito alto.

Método MNLPC	*Setup*	Objetos	Desp.	Exc.
SHP	12	544	14.16	0
MTB2	5	644	3.42	24.06
B&B	5	584	4.88	15.15
GCN	5	644	3.42	24.06
GCL	6	574	4.17	14.47
Solução	5	644	3.08	24.40

Tabela 3.6: Resultados para o problema da Classe 8, para $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

Os padrões gerados, com seu respectivo desperdício t, e a solução final, com os padrões gerados, são mostrados na Tabela 3.7. Na solução final, os padrões a_j, com $x_j > 0$, foram a_2 e a_9, gerados por SHP, a_{15}, gerado por MTB2, a_{21}, gerado por GCN, a_{22} e a_{23}, gerado por GCL. Note que os padrões a_{14}, a_{17} e a_{20}, e, a_{13}, a_{19} e a_{22} são idênticos. Por isso, apesar de não aparecer na solução final, podemos dizer que os padrões a_{13} e a_{22}, gerados por MTB2 e GCL, respectivamente, entraram na solução já que é o mesmo padrão, a_{19}, gerado por GCN. Portanto, para este problema, entraram na solução final 2 padrões de SHP, 1 padrão de MTB2, 1 padrão de GCN (ou MTB2 ou GCL) e 1 padrão de GCL.

3.2.2 Classe 9

Nesta subseção realizamos experimentos numéricos, análogos à subseção anterior, para um número maior de problemas: 10 problemas da Classe 9.

Para cada problema N, $N = 1, \ldots, 10$, resolvemos cada problema através dos 5 diferentes esquemas de geração de padrões: SHP e o Algoritmo MNLPC com MTB2, B&B, GCN e GCL. Fixamos $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$. Os resultados obtidos estão mostrados na Tabela 3.8.

	SHP											
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
	0	**1**	1	1	0	0	0	0	**0**	0	0	0
	0	**0**	0	0	0	0	0	0	**0**	0	1	0
	1	**0**	0	0	1	0	0	0	**1**	1	0	0
	0	**0**	0	0	0	0	1	0	**0**	0	0	0
	0	**0**	0	0	0	1	1	2	**0**	0	0	1
	0	**0**	0	0	0	2	0	0	**1**	0	0	0
	0	**0**	0	0	1	0	0	0	**0**	0	0	0
	2	**2**	1	0	0	0	0	0	**0**	0	0	0
	0	**0**	0	0	0	0	0	0	**0**	0	0	0
t	0.051	0.024	0.057	0.206	0.068	0.016	0.158	0.248	0.045	0.349	0.239	0.624
Solução	0	**220**	0	0	0	0	0	0	**237**	0	0	0

	MTB2			B&B			GCN			GCL	
	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}	a_{23}
	0	0	**0**	0	0	0	0	0	**0**	0	**0**
	1	0	**0**	1	0	0	1	0	**0**	1	**0**
	0	0	**0**	0	0	0	0	0	**0**	0	**0**
	0	1	**1**	0	1	0	0	1	**0**	0	**0**
	0	1	**0**	0	1	0	0	1	**0**	0	**2**
	0	0	**0**	0	0	0	0	0	**0**	0	**0**
	0	0	**1**	0	0	0	0	0	**0**	0	**0**
	1	1	**1**	0	1	0	1	1	**6**	1	**1**
	0	0	**1**	2	0	10	0	0	**1**	0	**1**
	0.090	0.009	0.013	0.057	0.009	0.090	0.090	0.009	0.015	0.090	0.008
	0	0	**115**	0	0	0	0	0	**22**	0	**50**

Tabela 3.7: Padrões gerados para o problema da Classe 8, para $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

A Tabela 3.9 mostra os resultados obtidos. A sigla `e` significa quantos dos `g` padrões gerados pelos diferentes métodos entraram na solução final, ou seja, a solução quando o problema é resolvido com todos os padrões gerados. Pode ocorrer a geração de um mesmo padrão por dois ou mais diferentes métodos. Como padrões gerados semelhantes não foram retirados, nos problemas 3, 5, 6, 8 e 10, o número entre parênteses é o número de padrões

	SHP			MTB2			B&B			GCN			GCL		
N	*Setup*	Objetos	Desp.	*Setup*	Objetos	Desp.	*Setup*	Objetos	Desp.	*Setup*	Objetos	Desp.	*Setup*	Objetos	Desp.
1	20	98	11.94	11	104	2.09	11	102	4.53	11	102	4.53	11	101	4.36
2	22	82	10.71	9	96	6.56	11	88	3.89	11	88	3.89	11	88	3.89
3	18	161	28.18	13	161	6.88	13	161	0.06	13	161	3.46	13	161	14.47
4	15	75	4.06	9	82	1.64	10	79	3.07	10	79	3.07	10	79	3.07
5	22	96	6.63	12	121	2.61	12	104	1.52	12	104	1.95	12	107	2.54
6	19	116	16.83	11	116	2.99	11	121	2.59	11	117	8.12	11	117	8.12
7	20	88	10.32	8	107	3.35	9	98	3.70	8	99	3.59	9	98	3.70
8	21	124	17.70	12	132	5.64	12	132	4.93	12	132	4.06	12	134	6.54
9	19	98	16.99	9	98	2.62	9	108	1.83	10	102	3.04	9	98	1.81
10	21	107	19.45	12	115	8.58	14	112	11.05	13	111	8.57	14	112	11.05

Tabela 3.8: Resultados para os problemas da Classe 9 para $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

que não entraram na solução final, porém são os mesmos gerados por outro método e estes, por sua vez, fazem parte da solução.

	Solução			SHP		MTB2		B&B		GCN		GCL	
N	*Setup*	Objetos	Desp.	e	g	e	g	e	g	e	g	e	g
1	10	104	1.93	7	20	2	5	0	2	0	2	1	3
2	9	96	5.71	6	22	3	6	0	1	0	1	0	1
3	13	161	0.06	0	18	1	7	12(1)	15	0(1)	10	0	3
4	9	82	1.64	7	15	2	3	0	1	0	1	0	1
5	12	104	1.52	10	22	0	2	2	3	0(1)	3	0	3
6	11	116	2.65	7	19	1(1)	5	3	5	0	1	0	1
7	8	107	2.07	6	20	1	1	0	2	1	2	0	2
8	12	132	3.97	7	21	1(1)	6	0(1)	5	3	6	1(1)	4
9	9	98	1.65	7	19	1	3	0	3	0	5	1	3
10	12	115	8.58	11	21	1	2	0	2	0(1)	6	0	2

Tabela 3.9: Resultados para problemas da Classe 9 para $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

Analisando a Tabela 3.9 dos padrões gerados, temos que MTB2 gerou pelo menos 1 padrão que entrou na solução final para quase todos os problemas, exceto para $N = 5$, B&B para 4 problemas, GCN para 5 e GCL para 3. A eficiência da geração de padrões de MTB2 foi 15/40, ou seja, 15 colunas das 40 geradas entraram na solução final, de GCN foi 7/37 e de GCL foi 3/23. O Método B&B teve a melhor eficiência, 19/39, ou seja, quase metade dos padrões gerados entraram na solução.

3.3 Comparação com ANLCP100

A fim de comparar o Método MNLPC com o ANLCP100, executamos o algoritmo implementado por Salles Neto, [20] na mesma máquina utilizada para a execução dos nossos testes numéricos. A versão do Algoritmo MNLPC que melhor se aproxima do ANLCP100 é com MTB2 para geração de colunas e fixando os parâmetros $c_1 = 1$, $c_2 = 100$ e $c_3 = 0$. Para cada uma das classes de problemas de CUTGEN1 foram geradas 100 instâncias. A Tabela 3.10 mostra os resultados obtidos.

A Figura 3.4 mostra a variação do *setup* e do número de objetos processados do MTB2 ($c_1 = 1$, $c_2 = 100$ e $c_3 = 0$) com relação a ANLCP100. Observamos que, comparado

Classe	ANLCP100					Método MNLPC		
	SHP		ANLCP100			MTB2		
	Setup	Objetos	*Setup*	Objetos	Tempo(s)	*Setup*	Objetos	Tempo(s)
1	4.01	14.21	3.40	14.55	0.33110	2.78	17.60	0.21617
2	5.92	116.66	4.86	118.82	0.77041	3.09	146.83	0.26201
3	5.00	25.15	4.92	24.19	0.19393	4.44	31.15	0.31874
4	7.32	224.29	7.37	223.69	0.23657	4.52	270.12	0.38462
5	6.96	47.01	7.06	45.94	0.44247	7.10	60.37	0.48639
6	10.87	434.18	10.85	433.28	0.57748	7.72	509.11	0.52707
Média	6.68	143.58	6.41	143.41	0.42533	4.94	172.53	0.36583
7	8.81	56.80	5.66	54.95	8.55037	5.76	55.64	0.08565
8	9.74	522.77	8.25	517.80	4.14450	5.86	551.26	0.08889
9	16.98	107.04	11.60	109.28	46.34606	10.88	105.68	0.17697
10	19.23	984.62	14.44	973.71	63.47121	11.26	1025.24	0.17713
11	32.37	202.77	22.03	226.61	119.93005	20.54	201.47	0.43587
12	37.68	1879.92	26.59	1881.18	281.42078	21.83	1944.51	0.47031
Média	20.80	625.65	14.76	627.25	87.31049	12.69	647.30	0.23914
13	9.39	71.22	6.12	65.15	3.90398	6.94	67.36	0.06308
14	9.86	653.55	7.97	617.88	3.34430	7.01	669.86	0.06404
15	18.00	134.61	9.71	114.41	29.23617	13.02	127.45	0.11353
16	19.54	1239.43	14.69	1161.40	27.51547	13.29	1265.46	0.12325
17	34.80	256.38	23.47	263.62	138.83259	24.47	242.77	0.27466
18	38.25	2376.84	31.04	2391.97	293.82523	25.42	2404.55	0.33170
Média	21.64	788.67	15.50	769.07	82.77629	15.02	796.24	0.16171

Tabela 3.10: Médias de ANLCP100 e MTB2 para $c_1 = 1$, $c_2 = 100$ e $c_3 = 0$.

ao ANLCP100, o Método MNLPC obteve soluções com menor número de *setup* em 13 classes. Entretanto, o número de objetos processados foi maior em 15 classes. Com relação ao tempo de execução, o Método MNLPC resolveu os problemas em menor tempo em 14 classes. Apenas nas classes 3, 4 e 5 o tempo gasto foi ligeiramente maior. O Método MNLPC conseguiu, para todas as classes, média de tempo de execução sempre menor que 0.6 segundos. Já ANLCP100, em algumas classes de problemas, a média de tempo de execução foi maior que 100 segundos. Na classe 18, por exemplo, a média do tempo de execução, em segundos, do ANLCP100 foi de 293.82523, enquanto que para o MTB2 foi de 0.33170.

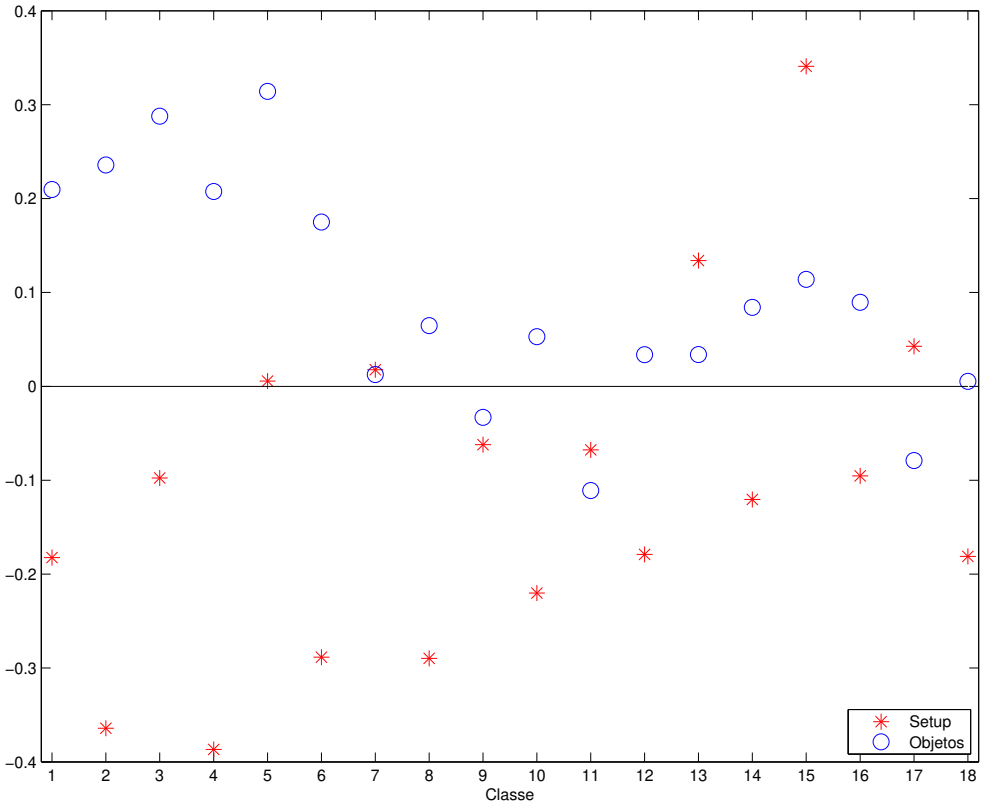


Figura 3.4: Variação de MTB2 em relação a ANLCP100.

3.4 Testes utilizando *Simulated Annealing*

Ao invés de utilizarmos os chutes aleatórios como estratégia de globalização, utilizamos a Heurística *Simulated Annealing* para o Algoritmo MNLPC com MTB2. Realizamos os testes com a forma irrestrita e a restrita. Ambas tiveram um desempenho semelhante, portanto mostramos os resultados para uma delas. A Tabela 3.11 mostra os resultados obtidos para a forma restrita.

Classe	*Setup*	Objetos	Desp.	Exc.	Tempo(s)
1	2.37	20.44	0.03	44.49	0.05852
2	2.46	204.42	0.01	43.07	0.04700
3	3.68	39.97	0.00	45.33	0.13901
4	4.07	368.97	0.00	40.67	0.11941
5	6.61	83.67	0.00	48.40	0.16365
6	6.99	808.38	0.00	46.81	0.19921
Média	4.36	254.31	0.01	44.80	0.12113
7	6.05	57.52	2.85	22.87	0.06493
8	6.14	566.69	2.75	21.80	0.06524
9	11.45	117.20	0.64	28.07	0.41871
10	11.60	1159.10	0.64	27.30	0.48995
11	20.84	235.08	0.05	30.11	2.62352
12	21.24	2353.00	0.05	30.19	2.98143
Média	12.89	748.10	1.16	26.72	1.10730
13	7.20	72.17	7.14	20.81	0.05256
14	7.28	719.21	7.14	20.56	0.06652
15	13.97	139.76	3.64	23.17	0.30242
16	14.17	1404.98	3.59	23.71	0.40690
17	27.16	279.38	1.66	26.01	2.52320
18	27.27	2784.87	1.68	25.84	3.18552
Média	16.18	900.06	4.14	23.35	1.08952

Tabela 3.11: Médias para MTB2, $c_1 = 1$, $c_2 = 10$, $c_3 = 100$, utilizando *Simulated Annealing*.

Na Figura 3.5, as variações dos tempos de execução para as classes 11, 12, 17 e 18 não foram mostrados porque foram, respectivamente, 4.04 e 3.68, 2.77 e 3.07. Para estas classes o tempo de execução utilizando *Simulated Annealing* foi muito alto. Para as classes com

itens pequenos, a estratégia que teve menor tempo foi o *Simulated Annealing*. Em relação ao número de objetos processados, a estratégia de gerar chutes aleatórios foi superior para todas as classes. Para as classes de itens pequenos, o *Simulated Annealing* obteve *setup* menor que os chutes aleatórios. Para as outras classes, a estratégia com chutes aleatórios foi superior com relação ao *setup*. A estratégia do *Simulated Annealing* se mostrou mais eficiente com relação ao desperdício, já que obteve menor percentagem para todas as classes. Acreditamos que isto se deve ao fato de ser mais difícil conseguir uma mudança em relação ao *setup*, isto é, da variável entrar ou não na solução, e que, fixado o número de variáveis não-nulas, a heurística procura melhorar o desperdício. Por outro lado, o excesso foi maior também para todas as classes.

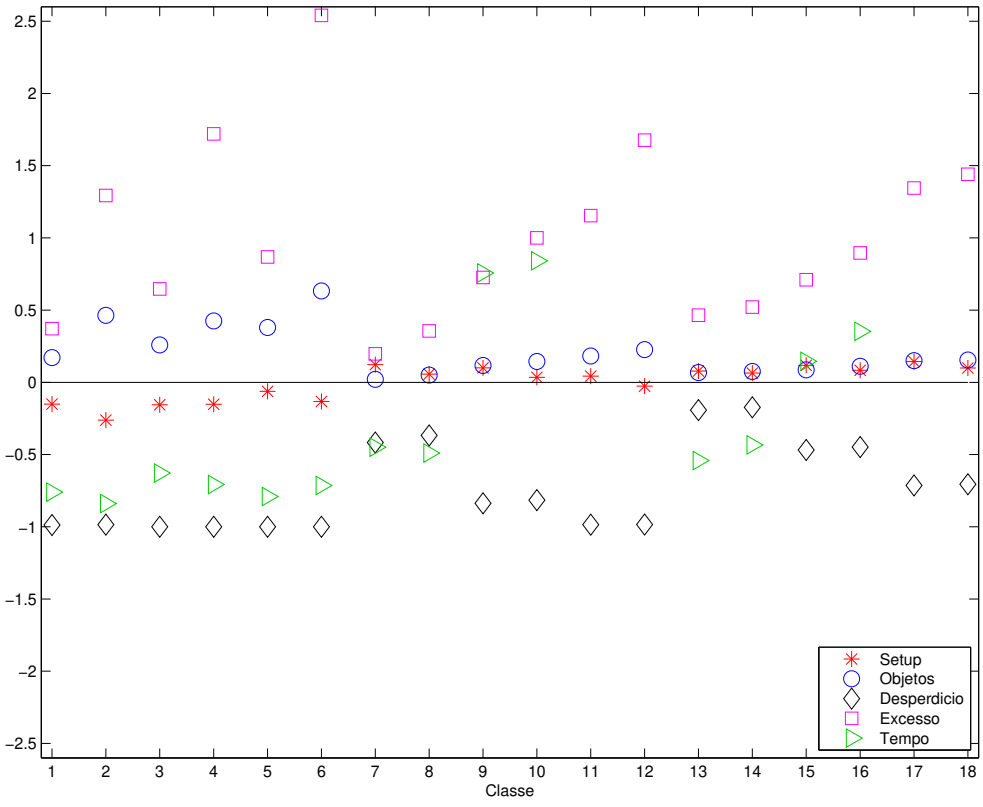


Figura 3.5: Variação de MTB2 com *Simulated Annealing* em relação com chutes aleatórios.

3.5 Teste com custo de *setup* dinâmico

Nesta subseção realizamos experimentos numéricos com a proposta de custo de *setup* (c_2) dinâmico para MTB2, com $c_1 = 1$ e $c_3 = 10$ fixos. A Tabela 3.12 mostra os resultados obtidos. A comparação desta versão é feita com MTB2 com $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$, ver Seção 3.2.

Classe	*Setup*	Objetos	Desp.	Exc.	Tempo(s)
1	4.54	12.92	0.72	14.35	0.17109
2	4.28	116.71	0.58	5.32	0.25274
3	5.96	23.54	0.68	7.56	0.31434
4	5.72	228.89	0.61	5.15	0.39790
5	8.96	45.14	0.67	5.08	0.88662
6	9.35	447.62	0.70	4.47	0.81557
Média	6.47	145.80	0.66	6.99	0.47304
7	5.99	52.85	4.34	14.02	0.11637
8	6.10	526.31	4.38	13.64	0.12705
9	11.59	99.61	3.07	12.28	0.26378
10	11.59	999.26	3.35	12.50	0.27502
11	21.87	188.82	2.85	10.00	0.55335
12	22.35	1899.87	3.09	10.40	0.65752
Média	13.25	627.79	3.51	12.14	0.33218
13	6.94	66.37	8.58	12.84	0.11969
14	7.02	662.24	8.55	12.73	0.12637
15	13.05	125.76	6.20	12.18	0.29986
16	13.25	1261.18	6.21	12.56	0.32874
17	24.58	238.66	5.07	10.25	0.76433
18	25.11	2394.69	5.25	10.43	0.90914
Média	14.99	791.48	6.64	11.83	0.42469

Tabela 3.12: Médias para MTB2 com custo de *setup* (c_2) dinâmico.

A Figura 3.6 mostra a variação do *setup*, número de objetos processados, desperdício e excesso de produção de MTB2 ($c_1 = 1$ e $c_3 = 10$) com $c_2 = obj_{SHP}$ em relação a $c_2 = 100$. Observamos que para as classes com itens pequenos, MTB2 com parâmetro c_2 dinâmico foi inferior em relação ao *setup*, entretanto foi superior em relação às demais variáveis. Para as demais classes, este comportamento se repete, porém as variações são menores.

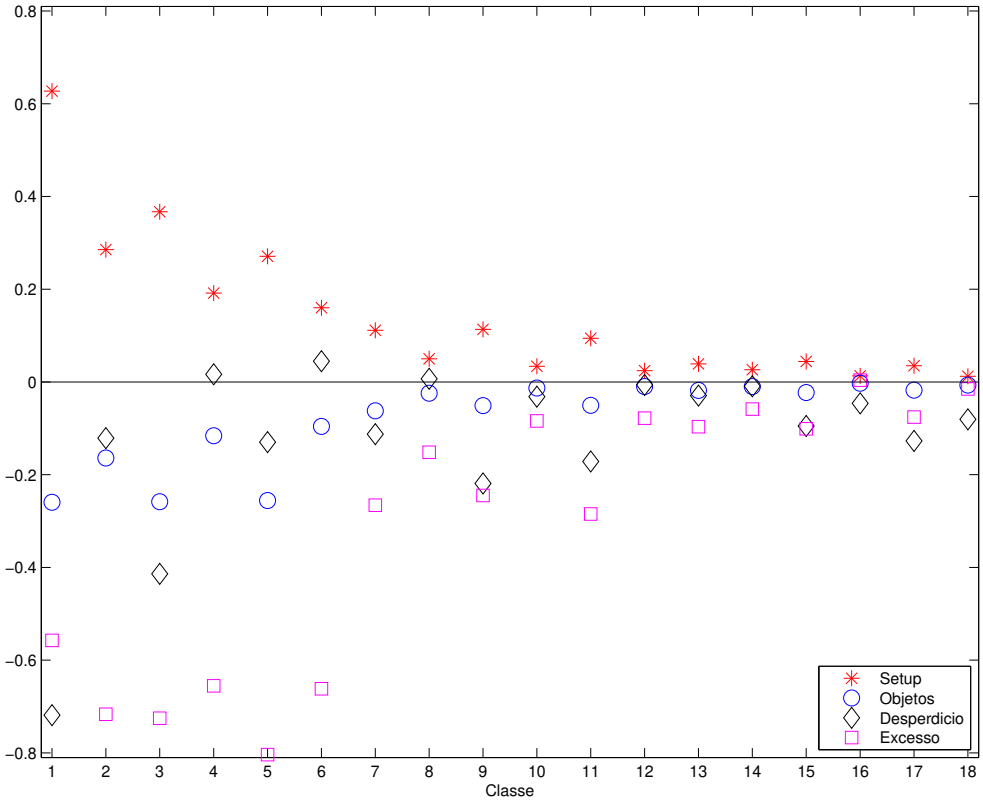


Figura 3.6: Variação de MTB2 utilizando $c_1 = 1$, $c_2 = obj_{shp}/n$ e $c_3 = 10$ em relação a $c_1 = 1$, $c_2 = 100$ e $c_3 = 10$.

Capítulo 4

Considerações finais e trabalhos futuros

Neste trabalho, propusemos uma formulação do problema de corte com os objetivos de minimizar o número de objetos processados, o *setup* e o desperdício. Estes objetivos são conflitantes e não conseguimos encontrar nenhum trabalho com os resultados para os problemas de CUTGEN1 que aborde assim esses três objetivos. Umetani, Yagiura e Ibaraki [21], apresentaram os resultados de seu método ILS-APG apenas em relação ao *setup* e ao desperdício. Os mesmos autores, [22], apresentaram os resultados de seu método modificado, ILS, em relação ao número de objetos e ao *setup*. Além disso, estes métodos trabalham minimizando o desvio padrão da demanda, enquanto que o método que apresentamos neste trabalho atende a demanda, mas há excesso de produção, ficando difícil fazer uma comparação entre eles.

Implementamos um método, baseado no ANLCP100, proposta de Salles Neto [20], que é mais rápido. O estudo aqui realizado a respeito dos multiplicadores de Lagrange do problema não-linear P_k e do problema auxiliar PLA2 foi importante para que a etapa de resolução de um problema linear auxiliar para se obter as variáveis duais pudesse ser evitada.

Propusemos um formulação não-linear para a geração de padrões, que obteve resultados satisfatórios, e, principalmente apontou que novas propostas para a geração de padrões podem ser competitivas.

Apesar de não obtermos resultados muito satisfatórios em relação ao *Simulated Anne-*

aling, o uso de estratégias de globalização deve ser tema de mais pesquisa, já que estamos procurando a solução global do problema e com os chutes aleatórios necessitamos resolver muitas vezes um mesmo problema, que dependendo do método utilizado, pode ser muito demorado.

A proposta de se utilizar o custo de *setup*, c_2, como um parâmetro dinâmico mostrou-se promissora. Obtivemos resultados bons em relação ao parâmetro não-linear fixo em 100, já que com esta proposta o número de objetos processados, o desperdício e o excesso de produção foram menores, apesar do *setup* ter sido maior. Portanto, a proposta é válida no intuito de minimizar o *setup* mas sem aumentar demasiadamente o número de objetos processados.

Referências Bibliográficas

[1] Aarts, E. H. L., Eiben, A. E. e Van Hee, K. M., "A general theory of genetic algorithms", Technical Report, Eindhoven University of Technology, Department of Mathematics and Computer Science, Holanda, 1989.

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

[4] Chelouah, R. e Siarry, P., "Tabu search applied to global optimization", *European Journal of Operational Research*, 2000.

[5] Chvátal, V., *Linear Programming*, Freeman, New York, 1983.

[6] Diegel, A. , Montocchio, E., Walters, E., Van Schalkwyk, S. e Naidoo, S.,"Setup minimising conditions in the trim loss problem", *European Journal of Operational Research*, 95, pp. 631-640,1996.

[7] Friedlander, A. , Martínez, J. M. e Santos, S. A., "A new trust region algorithm for bound constrained minimization", *Appl. Math. and Optim.* 30, pp. 235-266, 1994.

[8] 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.

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

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

[11] Glover, F., "Tabu search: Part I", *ORSA Journal on Computing*, 1989.

[12] Glover, F., "Tabu search: Part II", *ORSA Journal on Computing*, 1990.

[13] Haeser, G. "Algoritmo duas fases em otimização global", Dissertação de Mestrado, IMECC, UNICAMP, Campinas/SP, 2006.

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

[15] Kirkpatrick, S., Gelatt, C. D. e Vecchi, M.P., "Optimization by simulated annealing", *Science*, 220, pp. 671-680, 1983.

[16] Martello, S., e Toth, P., *Knapsack problems: algorithms and computer implementations*, John Wiley & Sons, 1990.

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

[18] Murtagh, B. A. e Saunders, M. A., MINOS 5.4 USER'S GUIDE (revised), Technical Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA 94305, USA, 1993. Revised 1995.

[19] Siarry, P., Berthiau, G. , Durbin, F. e Haussy, J., "Enhanced simulated annealing for globally minimizing functions of many continuous variables", *ACM Transactions on Mathematical Software*, 23(2), pp. 209-228, 1997.

[20] Salles Neto, L. L., "Modelo não-linear para minimizar o número de objetos processados e o setup num problema de corte unidimensional", Tese de Doutorado, IMECC, UNICAMP, Campinas/SP, 2005.

[21] Umetani, S., Yagiura, M. e Ibaraki, T., "One-dimensional cutting stock problem to minimize the number of different patterns", *European Journal of Operational Research*, 166(2), pp. 388-402, 2003.

[22] Umetani, S., Yagiura, M. e Ibaraki, T., "One-dimensional cutting stock problem with given number of setups: a hybrid approach of metaheuristics and linear programming", *Journal of Mathematical Modelling and Algorithms*, 5, pp. 43-64, 2006.

[23] Vanderbeck, F., "Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem", *Operations Research*, 48, pp. 915-926, 2000.

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

[25] Yanasse, H. H. e Limeira, M. S., "A hybrid heuristic to reduce the number of different patterns in cutting stock problems", *Comput. Oper. Res.*, 33(9), pp. 2744-2756, 2006.