UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA

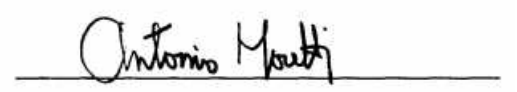
JULIANO DA SILVA DE SOUZA

UMA APLICAÇÃO SIMULATED ANNEALING EM PROBLEMAS DE CORTE DE ESTOQUE

DISSERTAÇÃO de MESTRADO APRESENTADA AO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA DA UNICAMP PARA OBTENÇÃO DO TÍTULO DE MESTRE EM MATEMÁTICA APLICADA.

ORIENTADOR: PROF.DR. ANTONIO CARLOS MORETTI

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO **JULIANO DA SILVA DE SOUZA** E ORIENTADA PELO **PROF.DR. ANTONIO CARLOS MORETTI**.



Assinatura do Orientador

Campinas, 2012.

FICHA CATALOGRÁFICA ELABORADA POR ANA REGINA MACHADO – CRB8/5467
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA – UNICAMP

So89a

Souza, Juliano da Silva de, 1984-
Uma aplicação simulated annealing em problemas de corte de estoque / Juliano da Silva de Souza. – Campinas, SP : [s.n.], 2012.

Orientador: Antonio Carlos Moretti.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Simulated annealing (Matemática). 2. Problema do corte de estoque. 3. Pesquisa operacional. I. Moretti, Antonio Carlos, 1958-. II. Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica. III. Título.

Informações para Biblioteca Digital

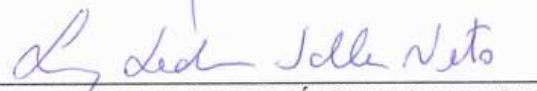
Título em inglês: A simulated annealing application for cutting stock problem
Palavras-chave em inglês:
Simulated annealing (Mathematics)
Cutting stock problem
Operational research
Área de concentração: Matemática Aplicada
Titulação: Mestre em Matemática Aplicada
Banca examinadora:
Antonio Carlos Moretti [Orientador]
Cristiano Torezzan
Luiz Leduíno de Salles Neto
Data da defesa: 13-02-2012
Programa de Pós-Graduação: Matemática Aplicada

Dissertação de Mestrado defendida em 13 de fevereiro de 2012 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.

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

Prof.(a). Dr(a). CRISTIANO TOREZZAN



Prof.(a). Dr(a). LUIZ LEDUÍNO DE SALLES NETO

Para Marisa e Valdir.

"Se a posse de um mundo ganhaste
não te alegres por isso, não importa...
Se a posse de um mundo perdeste
não sofras por isso, não importa...
Passam-se as dores e os prazeres...
E, tu, pelo mundo passas, não importa."
O livro da natureza.

Agradecimentos

Agradeço aos meus pais, Marisa e Valdir, pelo apoio e confiança que sempre dedicaram às minhas idealizações.

Agradeço também aos meus irmãos, Márcia, Júnior e Renato, pelos incentivos e apoio em quedas.

Agradeço aos amigos e colegas Ana Camila, Carolina, Eliel, Fábio, Júnior, Mael, em especial agradeço pela parceria e companheirismo ao meu grande amigo Felipe, pela força e generosidade meu imenso obrigado ao Rafael, pelo apoio e ajuda em um dos momentos mais difíceis agradeço ao Márcio e aos amigos da H2, e, por grandes estudos e discussões produtivas agradeço ao Clayton. Aos amigos Mateus, Maicon e Valmício pela força mandada de Santa Catarina.

Um agradecimento especial ao meu orientador Moretti pela amizade permitida, e pela oportunidade do crescimento profissional e pessoal. Também agradeço ao professor Leduíno por disponibilizar as classes geradas em sua tese de doutorado.

Agradeço ao professor Cristiano pela participação na banca e a todos os professores do IMECC que contribuíram para que este trabalho se realizasse.

À CAPES pelo apoio financeiro.

Agradeço a todos que de uma maneira ou de outra me auxiliaram a transformar o sonho em uma realidade e que não nominei aqui. Sou eternamente devedor à todos que tornaram o caminho e as pedras mais leves e mais alegre.

Resumo

Neste trabalho é apresentada uma nova abordagem da heurística Simulated Annealing, no que se refere a geração de soluções na vizinhança de uma solução factível, para encontrar a solução ótima de uma formulação de programação linear inteira para o Problema de Corte de Estoque Unidimensional. O desempenho do novo algoritmo é comparado à metodologia publicada em *A simulated annealing heuristic for the one-dimensional cutting stock problem* apresentada em [2]. Os resultados dos experimentos computacionais indicam que essa nova abordagem, fornece soluções muito melhores em relação ao valor objetivo em tempo equivalente de execução. Além disso, uma comparação qualitativa é feita com o *solver CPLEX*. Para os experimentos numéricos utiliza-se o gerador de problemas *CUTGEN1: A problem generator for the Standard One-dimensional Cutting Stock Problem*, proposto em [6], o qual fornece um gerador de classes de problemas de acordo com os critérios de tamanho dos itens finais e demandas. Finalmente, são reportados resultados dos experimentos computacionais baseados na metodologia apresentada em [1] no artigo *Guidelines for Designing and Reporting on Computational Experiments with Heuristic Methods*.

Abstract

This work presents a new approach to heuristic Simulated Annealing, in refers to the generation of solutions in the neighborhood of a feasible solution, to find the solution an optimal integer linear programming formulation for the Cutting Stock Problem One-dimensional. The performance of the new algorithm is compared to the methodology published in *A simulated annealing heuristic for the one-dimensional cutting stock problem* presented in [2]. The results of computational experiments indicate that this new approach provides much better solutions in relation to the objective value time equivalent execution. In addition, a qualitative comparison is made to the CPLEX solver. For the numerical experiments we use the generator of problems *CUTGEN1: A problem generator for the Standard One-dimensional Cutting Stock Problem*, in [6], which provides a generator classes of problems according to criteria size and demands of end items. Finally, results of experiments are reported computer-based method presented in [1] by article *Guidelines for Designing and Reporting on Computational Experiments with Heuristic Methods.*

Lista de Tabelas

Lista de Figuras

Sumário

Capítulo 1

Introdução

O objetivo desse trabalho está focado em resolver problemas de corte unidimensional, mais especificamente, Problemas de Corte de Estoque Unidimensional, os quais podem ser escritos como problemas de programação linear inteira combinatorial e são pertencente a classe $\mathcal{NP}$ - completo . Na formulação apresentada são considerados custos de *setup*, número de objetos processados e excesso de produção. Baseado na proposta publicada em [2], a resolução de tais problemas é efetuada por meio da heurística *Simulated Annealing* com uma nova proposta de geração de soluções na vizinhança de uma dada solução corrente.

Inicialmente, a partir do que foi publicado em [16], apresenta-se uma metodologia capaz de gerar todos possíveis padrões de corte factíveis, e a partir da implementação e execução do método, foi possível obter a matriz A (cujas colunas representam padrões de cortes) usada na obtenção de uma solução inicial. As soluções iniciais foram fornecidas através da resolução dos problemas resolvidos por relaxação linear.

Problemas de corte de estoque apresentam uma estrutura relativamente simples no que se refere a modelagem do problema, entretanto, a forma de resolução exata pode se tornar uma tarefa árdua uma vez que se trata de um problema de programação linear inteira e pertencente a classe $\mathcal{NP}$ - completo, o que implica ser difícil de se resolver de maneira

exata. Por essa razão é que em grande parte das aplicações utiliza-se uma heurística como recurso para obter soluções ou soluções aproximadas para esses problemas.

Em 1996, Chen [2] propõe utilizar a heurística *Simulated Annealing* em problemas de corte de estoque, visto que facilmente se encontra outras soluções por meio de uma estratégia para gerar soluções vizinhas e, além disso, é possível escapar de sub-ótimos locais podendo encontrar soluções suficientemente boas em tempo razoável. Após um estudo do que foi proposto, apresenta-se uma variação da geração da vizinhança em relação a [2] ao se observar uma relevante insuficiência em resoluções de exemplares de maior porte.

Em geral, problemas de corte e empacotamento podem ser classificados de diversas maneiras, tais como Dyckhoff [3] apresenta organizando de acordo com dimensão, atribuição de tipos, sortimento de unidades pequenas e grandes. Dentro desse contexto, Gau e Wäscher [6] propõem um gerador de problemas de corte de estoque unidimensionais, e portanto, do modo mais neutro possível, foram gerados vinte exemplares num total de dezoito classes, assim, pôde-se estabelecer uma comparação justa entre os métodos.

O presente texto está organizado na seguinte ordem. No Capítulo 2 faz-se uma menção ao surgimento das formulações e em que meios tais problemas costumam surgir, em seguida, apresenta-se a tipologia proposta por Dyckhoff [3]. Após uma breve explanação sobre padrões de corte, são mostradas algumas das formulações básicas, e dentre essas, destaca-se a formulação usada neste trabalho. Na sequência, destaca-se o método de gerar os possíveis padrões de corte a partir de um item em estoque. O Capítulo 3 apresenta a heurística *Simulated Annealing* e detalha-se a estratégia de geração de vizinhança usada por Chen [2] e a nossa proposta nesta pesquisa. No Capítulo 4 constam os experimentos computacionais, no qual são fornecidas as informações básicas sobre as classes obtidas pelo gerador de problemas, e apresenta-se como foram efetuadas as seleções de parâmetros necessários para a execução do algoritmo.

Ao final, para efeito de comparação qualitativa, toma-se os mesmos exemplares nos quais fora experimentada a heurística com ambas estratégias, e aplica-se o solver comercial *CPLEX*, reportando nesse contexto análises entre soluções fornecidas. A análise de um ponto de vista geral, segue de acordo com alguns aspectos importantes e necessários segundo o que Barr et al. [1] apresentam como proposta de análise de um método heurístico.

Capítulo 2

Problemas de corte

2.1 Introdução

Deve-se a Kantorovich o pioneirismo sobre os trabalhos de perdas em problemas de corte, produzido em meados de 1939 e publicado somente em 1960 [10]. Posteriormente Paul e Walter (1954) [14], Metzger (1958, [13]) e Eilon (1960, [5]) mostraram formulações semelhantes, entretanto, seus métodos adequavam-se somente a problemas pequenos. Um grande avanço se obteve após os trabalhos de Gilmore e Gomory [7, 8], desde então muitos pesquisadores dedicaram atenção aos problemas de corte.

Os problemas de corte surgem frequentemente em indústrias, pela necessidade de cortar peças grandes em peças menores afim de suprir demandas de clientes. Em outras palavras, supondo que é requerido uma demanda de m itens de tamanhos (largura, comprimento, volume) distintos $w_i, (i = 1, \cdots, m)$, a partir de n peças maiores de tamanhos $L_j, (j = 1, \cdots, n)$, deseja-se cortar de melhor modo as peças grandes, sujeito a restrições de demandas, visando a minimização dos custos ou maximização de lucros de acordo com uma dada especificidade pertinente a algum problema de corte.

De outra forma, pode-se minimizar o número de peças usadas no processamento de

itens menores. Tais problemas são conhecidos como *problema de empacotamento* de m itens distintos $w_i, (i = 1, \cdots, m)$ com características específicas (largura, comprimento, volume) que devem ser colocados em um recipiente de capacidade L.

Visto que há uma grande variedade de problemas de corte, Dyckhoff [3] classificou sistematicamente quatro importantes características dos problemas de corte, as quais tem um papel fundamental na escolha e na complexidade de soluções aproximadas. Em cada subclasse, definida por cada umas dessas características, têm-se tipos específicos que por sua vez formam 96 combinações de problemas bem estruturados. Abaixo, com símbolos apropriados (cf. Dyckhoff), está a classificação dos problemas de corte e entende-se por objetos as peças maiores que são processadas em itens finais que são demandados.

(α) **Dimensão**

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

(β) **Tipo de Atribuição**

- (B) Todos os objetos (objetos que serão processados) e uma seleção de itens demandados (itens finais);
- (V) Uma seleção de objetos (objetos que serão processados) e todos os itens demandados (itens finais);

(γ) **Classificação dos Objetos**

- (O) Um único objeto;
- (I) Vários objetos de tamanhos iguais;

(D) Vários objetos de tamanhos distintos;

(δ) **Classificação dos Itens**

(F) Poucos itens de diferentes tamanhos;

(M) Vários itens de diferentes tamanhos;

(R) Vários itens com poucos tamanhos diferentes;

(C) Itens de tamanhos iguais;

Cada problema obtido pela combinação dessas características é representado por uma quádrupla da forma $\alpha/\beta/\gamma/\delta$, por exemplo, o tipo dado por 3/B/O/F denota todos os problemas tridimensionais (3), onde um objeto grande (O) deve ser empacotado com uma seleção de itens (B) de poucos itens de diferentes tamanhos (F). As tipologias de alguns problemas clássicos são apresentadas na Tabela 2.1.

Problemas	Tipologia
Problema de corte de estoque (geral)	1/ / /, 2/ / /, ou 3/ / /
Problema de corte de estoque (clássico)	1/V/I/R
Problema da mochila (clássico)	1/B/O/
Problema de carregamento de Bin's (clássico)	1/V/I/M
Problema de carregamento de Bin's bidimensional	2/V/I/M
Problema de carregamento de contêiner	3/V/I/ , ou 3/B/O/

Tabela 2.1: Tipologia de Dyckhoff

Em 2007, Wäscher et al. [17] propõem uma melhoria na tipologia apresentada por Dyckhoff [3], a qual é apresentada por:

- **Dimensionalidade**

  (1) Unidimensional;

  (2) Bidimensional;

  (3) Tridimensional;

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

- **Tipo de Atribuição**

(OM) Maximização de saída: seleciona-se itens de modo a maximizar o uso dos objetos;

(IM) Minimização de entrada: aloca-se todos itens de forma a minimizar a utilização do objeto;

- **Variedades de Objetos**

  (O) Um único objeto;

    (Oa) Com todas as dimensões fixas;

    (Oo) Com uma ou mais dimensões variáveis;

    (Om) Com várias dimensões variáveis.

  (SF) Vários objetos

    (Si) Iguais;

    (Sw) Pouco heterogêneos;

    (Ss) Muito heterogêneos;

- **Variedade de Itens**

  (IS) Idênticos;

  (W) Vários itens pouco heterogêneos;

  (S) Vários itens muito heterogêneos;

- **Formato de Itens**

  Atribui-se aos problemas que possuem duas ou três dimensões uma classificação ao formato dos itens:

  (R) Regulares;

  (I) Irregulares;

Dessa forma, por exemplo, o problema de corte bidimensional representado por $2/V/D/R$ na tipologia de Dyckhoff [3] é expressado por $2/IM/Sw/W$ na tipologia apresentado por Wäscher et al. [17].

Na seção seguinte será apresentado e formulado o problema de corte que será considerado neste trabalho, cujo foco é encontrar uma solução ótima ou aproximada utilizando uma heurística como metodologia.

2.2 Problemas de Cortes de Estoque Unidimensional

Problemas de corte de estoque (PCE) surgem, como foi citado anteriormente, no procedimento de corte de objetos (i.e., matéria prima), à disposição no estoque, em itens menores, os quais devem suprir as demandas requeridas pelos clientes. Uma modelagem pode ser obtida na forma de um problema de otimização linear inteira de grande porte, que por sua vez determina quais e de que forma os padrões de corte utilizados serão usados no plano de corte, ou seja, define como as peças maiores podem ser cortadas em peças menores a fim de minimizar custos. Rapidamente, cada padrão de corte é uma combinação de itens que devem ser cortados a partir de uma peça maior, por exemplo, dado uma peça de tamanho $L > w_1, w_2, w_3$, e supondo que $L = w_1 + w_2 + 2w_3$, onde não há desperdício, obtém-se o seguinte padrão de corte como mostra a Figura 2.1:

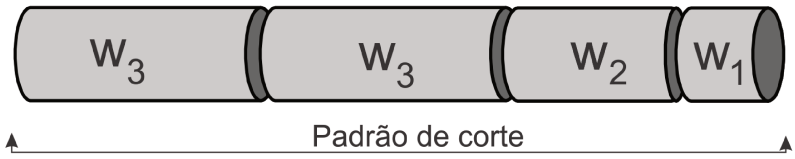


Figura 2.1: Padrão de corte em uma peça de largura L.

Em geral, um PCE pode ter como objetivo minimizar a quantidade de itens processados que pertencem ao estoque, a minimização das perdas totais num plano de corte, a minimização de número de *setup* de máquinas, ou a minimização de estoque intermediário, ou ainda, uma combinação desses objetivo, como por exemplo, a minimização da quantidade de itens processados mais *setup*, a qual será aqui abordada. Algumas dessas formulações serão abordadas na subseção seguinte.

2.2.1 Formulação do PCE Unidimensional

Como mencionado anteriormente, em um modelo no qual a minimização de peças processadas é o objetivo principal, deve-se obter itens menores a partir da escolha de padrões

de cortes de modo que a demanda seja atendida. Em outras palavras, dado um padrão j, e uma demanda d_i de um dado item i de tamanho w_i, denotamos por a_{ij} a quantidade do item i obtida ao se cortar o objeto mestre com o padrão de corte j, e por x_j a quantidade em que o objeto é cortado usando-se o padrão j. Logo, o modelo é dado por:

$$\begin{aligned} &\text{Minimizar} \quad C\sum_{j=1}^{n} x_j \\ &\text{Sujeito à} \quad \sum_{j=1}^{n} a_{ij}x_j = d_i\,, \quad i = 1, \cdots, m \\ &x_j \in \mathbb{N},\ j = 1, \cdots, n, \end{aligned}$$

em que C representa o custo de cada peça em estoque. Uma observação relevante é de que nesse modelo supõe-se que a demanda deve ser exatamente atendida.

Por outro lado, se o objetivo for minimizar o desperdício, então é necessário definir c_j como o desperdício obtido no padrão j, e então temos:

$$\begin{aligned} &\text{Minimizar} \quad C_1\sum_{j=1}^{n} c_j x_j \\ &\text{Sujeito à} \quad \sum_{j=1}^{n} a_{ij}x_j = d_i\,, \quad i = 1, \cdots, m \\ &x_j \in \mathbb{N},\ j = 1, \cdots, n, \end{aligned}$$

em que:

- C_1 é o custo por unidade desperdiçada;
- $c_j = L - \sum_{i=1}^{m} w_i a_{ij}\,, j = 1, \cdots, n$;

Ilustrativamente, a principal ideia pode ser mostrada na Figura 2.2.

É importante destacar que se as restrições de demanda forem de igualdade, o pro-

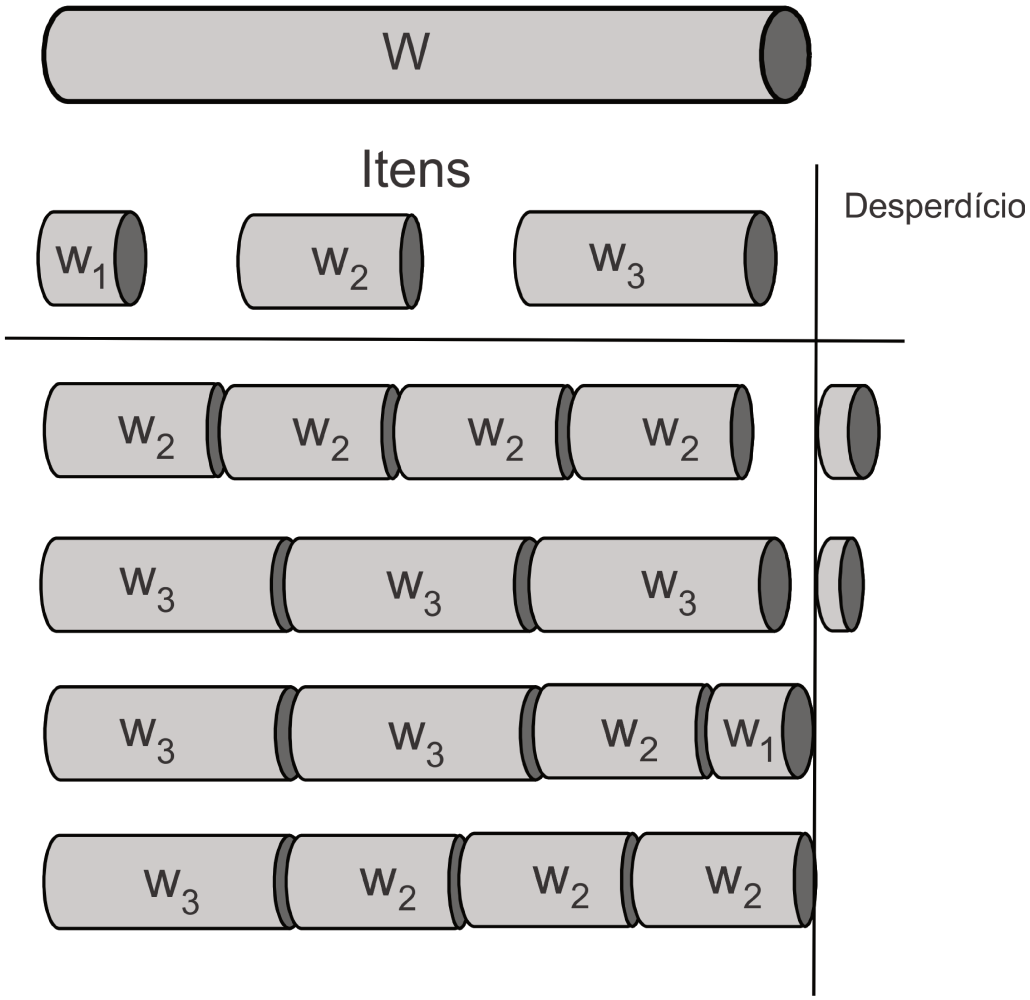


Figura 2.2: Cortes - Exemplo Ilustrativo

blema de minimizar a quantidade processada de itens em estoque é equivalente ao problema de minimizar o desperdício. De fato, supondo que as restrições são de igualdades, sabe-se que a um dado um padrão de corte j tem-se um desperdício associado que pode ser expressado por :

$$c_j = L - \sum_{i=1}^{m} a_{ij} w_i. \tag{2.1}$$

Facilmente, nota-se que o desperdício total obtido em uma dada solução factível é descrito por:

$$\sum_{j=1}^{n} c_j \, x_j = \tag{2.2}$$

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

Dessa forma, pela restrição de igualdade para a demanda, obtemos:

$$L \sum_{j=1}^{n} x_j - \underbrace{\sum_{i=1}^{m} w_i \, d_i}_{k} . \tag{2.4}$$

Portanto, uma vez que L, e k são constantes, pode-se concluir que os problemas de minimizar desperdício (2.2) e itens processados que pertencem ao estoque (2.4) são equivalentes.

Além de custos materiais, deve-se dar atenção ao tempo de produção, e portanto, em algumas indústrias torna-se necessário considerar o tempo para atender a demanda. Para esse modelo há uma mudança na função objetivo com o acréscimo de um custo fixo, tais custos são atribuídos ao *setup* (tempo e trabalho) necessário para arranjar as facas de corte na mudança de um padrão para outro. Vejamos:

$$\text{Minimizar} \quad C_1 \sum_{j=1}^{n} c_j x_j + C_2 \sum_{j=1}^{n} y_j$$

$$\text{Sujeito à} \sum_{j=1}^{n} a_{ij} x_j = d_i \, , \quad i = 1, \cdots, m$$

$$y_j \leq x_j \leq N y_j \, , \quad j = 1, \cdots, n$$

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

$$y_j \in \{0, 1\} \, , \quad j = 1, \cdots, n,$$

em que C_2 é o custo fixo por usar um padrão, e N é um número suficientemente grande.

Especificamente, o modelo que é foco desse trabalho tem como objetivo minimizar

a quantidade de itens processados pertencentes ao estoque e *setup* de produção, além disso, para uma expansão no espaço-solução faz-se com que a formulação permita uma folga na produção, ou seja, a formulação trabalhará com a desigualdade ("$\geq$") nas restrições de demandas permitindo assim um excesso, o qual será penalizado na função objetivo. A seguir, vejamos o modelo apresentado:

- **Parâmetros**
  - w_i = largura do item i;
  - a_{ij} = o número de unidades i cortadas com o padrão j da bobina mestre, $(i = 1, \cdots, m\,; \quad j = 1, \cdots, n\,)$.
  - d_i = demanda do item i;
  - C_1 = custo por cada item em estoque;
  - C_2 = custo fixo por *setup* de preparação da máquina para um padrão de corte;
  - M = número grande que penaliza cada unidade excedente;
  - N = número grande limitante para cada x_j;
- **Variáveis de Decisão**
  - x_j = o número de vezes que o padrão j é usado;
  - $y_j = \begin{cases} 1 & \text{, se o padrão } j \text{ é usado} \\ 0 & \text{, caso contrário} \end{cases}$
  - $sobra_i$ = a quantidade excedente a demanda do item i;
- **Modelo**

$$Minimizar \qquad C_1 \sum_{j=1}^{n} x_j + C_2 \sum_{j=1}^{n} y_j + M \sum_{i=1}^{m} sobra_i$$

$$sujeito \quad a: \qquad \sum_{j=1}^{n} a_{ij}\, x_j = d_i + sobra_i\,, \quad \forall i = 1, \cdots, m$$

$$y_j \leq x_j \leq N y_j, \forall j = 1, \cdots, n$$

$$x \in \mathbb{Z}_+^n, \quad sobra \in \mathbb{Z}_+^m, \quad y \in \mathbb{B}^n$$

Mesmo sendo problemas aparentemente simples, tem-se que devido ao grande número de possíveis padrões de corte e ao fato de ser um problema de Programação Linear Inteiro - PLI, o PCE unidimensional pertence a classe de problemas $\mathcal{NP}$-difíceis, o que significa que tal problema não é resolvido em tempo limitado por um polinômio em n. Portanto, é justificável que pesquisadores optem buscar heurísticas que fornecem soluções suficientemente boas. Em princípio, veremos a seguir, um método de padrões de corte que fornece todos os padrões possíveis, entretanto, para classes com alguma especificidade particular o método deve ser interrompido, pois a quantidade de padrões de corte chega a milhões, em contrapartida, mais tarde será proposto uma alternativa para tal percalço.

2.2.2 Geração de Padrões de Cortes

Suliman [16] apresenta a implementação do procedimento para gerar colunas, as quais representam todos os possíveis padrões de corte. Como já mencionado, em alguns casos não é possível gerar todos esses padrões de corte.

Por meio de um método de ramificação em árvore é possível encontrar todos os padrões de corte factíveis para o problema. Para cada objeto mestre k em estoque é construída uma árvore separada das outras, o nó inicial contém a largura do objeto mestre k (L_k) pertencente ao estoque e está localizado no que se denomina nível 1. Em cada ramo da árvore no nível $i\,, i = 1, \cdots, m$, é mostrado o produto entre o número de unidades de largura w_i que são cortadas no padrão j, e a largura requerida w_i, isto é, $a_{ijk}w_i$. Em um nível $i\,, i = 1, \cdots, m$, o nó armazena a largura restante após serem satisfeitos os cortes especificados pelos ramos

anteriores. Por fim, os últimos nós representam as perdas de material resultantes de cada padrão gerado.

Cada caminho percorrido da raiz (nó inicial) até o topo da árvore representa um padrão de corte factível, que por sua vez, forma uma coluna da matriz restrição, como pode ser visto na Figura 2.3.

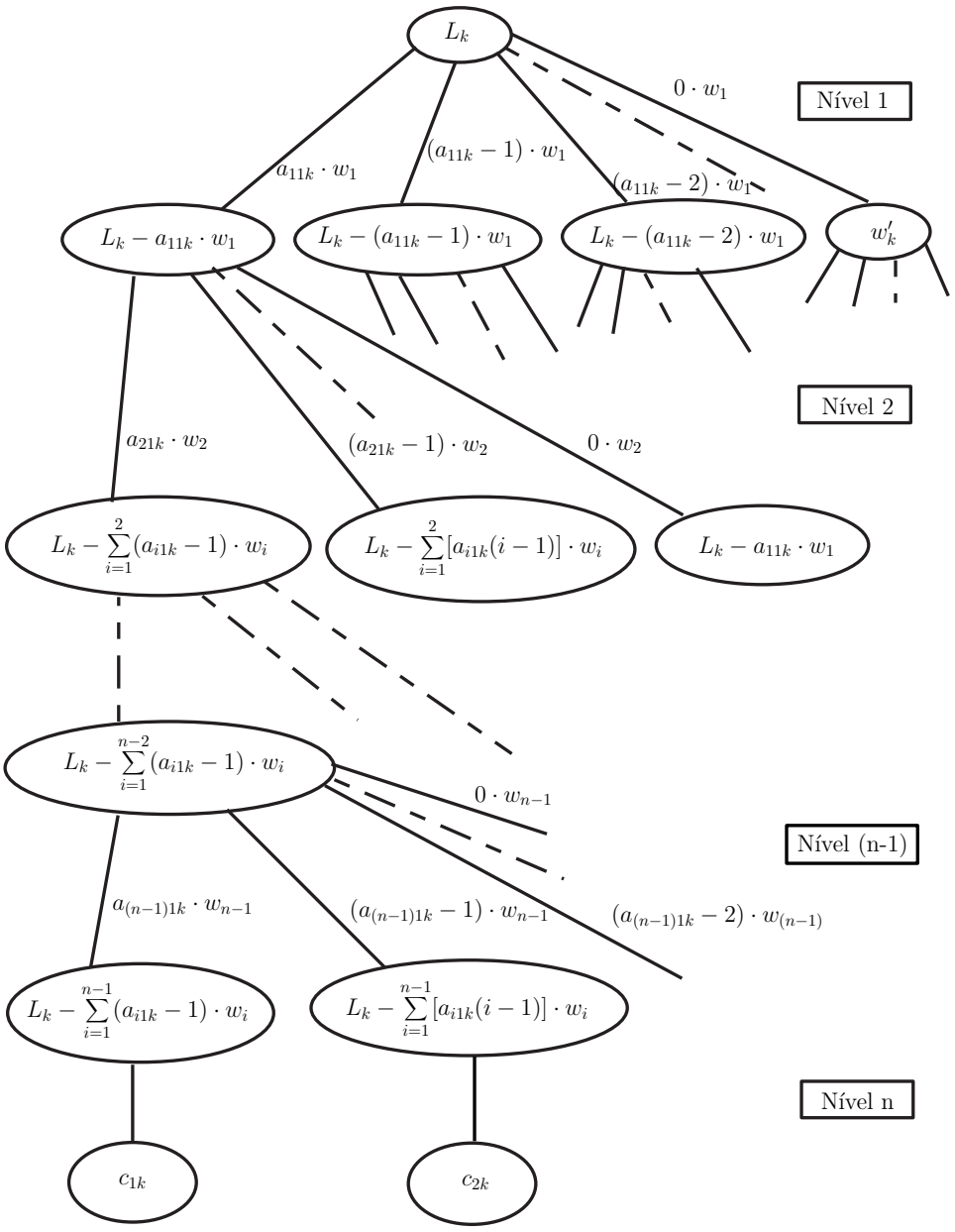


Figura 2.3: Geração de Padrões de Corte

Exemplo Ilustrativo

Supondo que existam objetos em estoque de um único tamanho $L = 100$ (onde $L = L_1$), e deseje-se cortar em itens finais de tamanhos $w_1 = 50, w_2 = 40, w_3 = 30$, e $w_4 = 20$, com uma dada demanda requerida. Então pode-se obter algumas configurações de cortes, como a Figura 2.4:

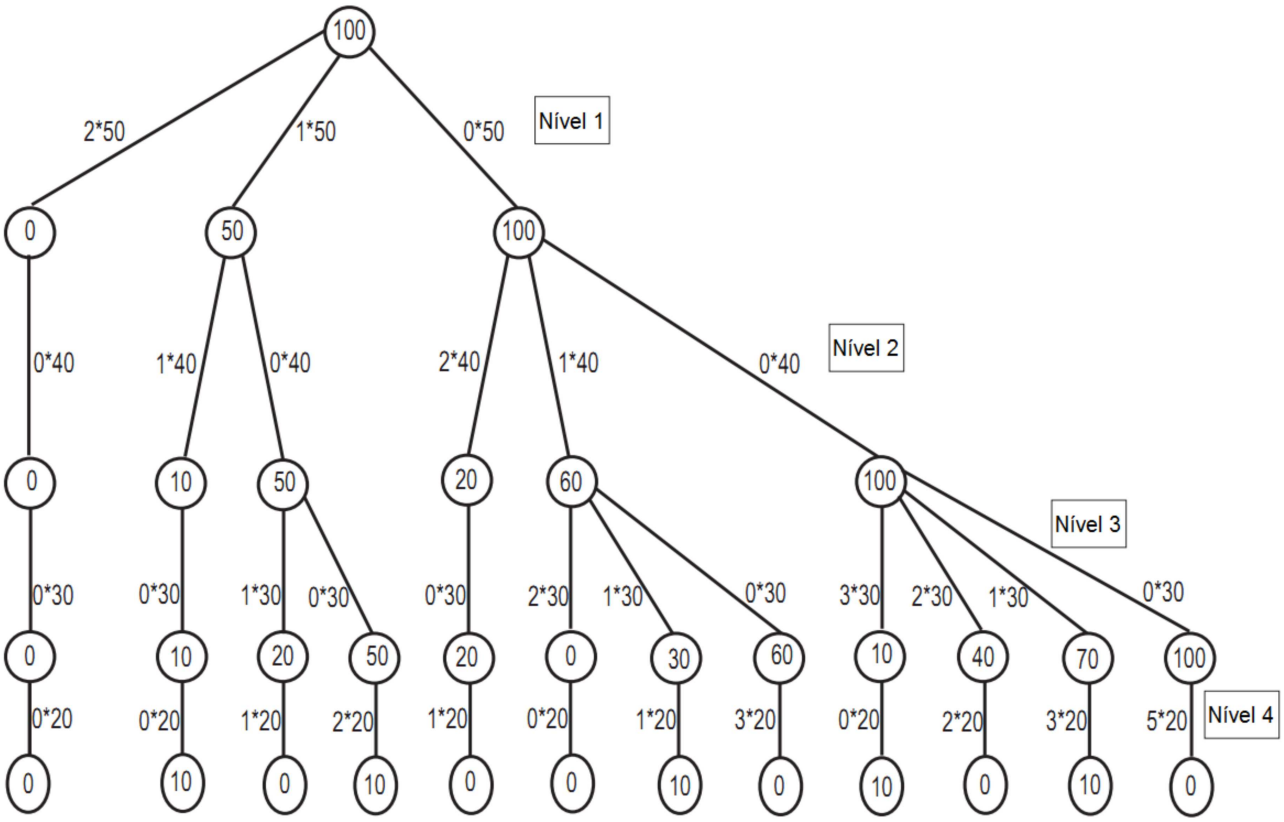


Figura 2.4: Geração de Padrões de Corte - Exemplo Ilustrativo

Portanto, para esse exemplo ilustrativo, obtém-se a matriz A e o vetor *desp* (desperdício), respectivamente:

$$A = \begin{bmatrix} 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 1 & 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 1 & 3 & 0 & 2 & 3 & 5 \end{bmatrix};$$

$$desp = \begin{bmatrix} 0 & 10 & 0 & 10 & 0 & 0 & 10 & 0 & 10 & 0 & 10 & 0 \end{bmatrix}.$$

Para a segunda parte da resolução do PCE, através da metodologia aqui empregada, tem-se a designação da solução inicial. Nesse ponto pode-se usar como solução inicial uma qualquer ou algo mais específico, desde que tal seja factível. Aqui, a solução usada é a obtida pela relaxação linear do problema, a qual após a obtenção, é arredondada para $x = \lceil x \rceil$ ($x = \lceil x \rceil$ é o menor inteiro maior ou igual a x)de tal modo que a demanda seja atendida, ou seja, factível.

Na formulação do problema relaxado não são considerados os custos de *setup*, pois basta que a solução seja factível, e portanto, as restrições que envolvem as variáveis binárias são eliminadas da formulação original do problema, isto é:

$$Minimizar \qquad C_1 \sum_{j=1}^{n} x_j + M \sum_{i=1}^{m} sobra_i$$

$$sujeito \quad a: \qquad \sum_{j=1}^{n} a_{ij}\, x_j = d_i + sobra_i\,, \quad \forall i = 1, \cdots, m$$

$$x \in \mathbb{Z}_+^n, \quad sobra \in \mathbb{Z}_+^m.$$

Dando continuidade, a proposta no próximo capítulo está baseada na idéia de usar uma heurística para encontrar o conjunto de padrões de corte que fornece a melhor solução para o PCE unidimensional visto que hipoteticamente foram obtidos todos os padrões de corte factíveis para efetuar a busca.

Capítulo 3

Simulated Annealing

3.1 Conceito Básico de Otimização Local

Dado um problema de otimização combinatorial o qual pode ser especificado por um conjunto de soluções juntamente com uma função custo que associa um valor a uma solução, define-se uma solução ótima como a solução que é associada ao menor custo possível. Assim, dada uma solução qualquer para um determinado problema, denomina-se por otimização local a tentativa de melhorar a solução por meio de um conjunto de ações, ou seja, melhorar localmente a solução atual. Essas ações são especificadas como pertubações na solução corrente, e geram o que se chama de soluções vizinhas, assim, dada uma solução s é possível obter soluções vizinhas a ela, isto é, soluções $s' \in \mathcal{N}(s)$ (Vizinhança de s). A partir disso, o algoritmo, de modo simples, toma a solução de menor custos e repete o procedimento. Ao final, encontra-se uma solução que não necessariamente é a ótima, mas tem o custo mínimo dentre todas as soluções em sua vizinhança.

Embora, em alguns problemas combinatoriais, seja possível encontrar uma boa solução, ainda que esses problemas pertençam a classe $\mathcal{NP}$-difícil, há grandes chances de que a solução obtida através da otimização local não seja boa o suficiente, pois dificilmente escapam

de locais sub-ótimos. Para contornar esse problema, pode-se aplicar a heurística *Simulated Annealing* (SA) desenvolvida em [11].

3.2 Simulated Annealing

A heurística *simulated annealing* destacou-se após ter sido apresentada no trabalho de *Kirkpatrick et al.* [11] que relacionou a simulação de resfriamento de sólidos proposta por Metropolis [12] com problemas de otimização. Posteriormente, surgiram pesquisas em problemas de otimização tais como processamento de imagens, física e química molecular, entre outros, (veja [4] e [9]). Como o próprio nome sugere, o *simulated annealing* é um procedimento computacional que simula o processo físico de resfriamento controlado em que o objetivo é levar o sistema ao seu estado mínimo de energia. Tal processo é divido em três fases:

(*i*) alta temperatura, na qual temos uma *configuração de alta energia*, por exemplo, em materiais físicos como o metal, onde há um alto aquecimento até a fundição do mesmo;

(*ii*) permanência em alta temperatura em que suas partículas se arranjam aleatoriamente;

(*iii*) diminuição de temperatura lentamente onde ocorre uma *configuração de energia* mais *baixa*.

Basicamente, a heurística se destaca por simular a aceitação de soluções que não são de melhora para que seja possível evitar soluções sub-ótimas, isto é, é possível que o algoritmo aceite uma solução de qualidade inferior de modo que a geração de soluções na vizinhança desta, possibilite a saída do local sub-ótimo. Exemplificando, suponha uma dada função objetivo $f(x)$ em um problema de minimização, uma solução inicial x_0, e uma solução $x' \in N(x_0)$ na vizinhança da solução inicial, a qual $f(x_0) < f(x')$, entretanto, supondo que a solução seja aceita, vejamos a Figura 3.1.

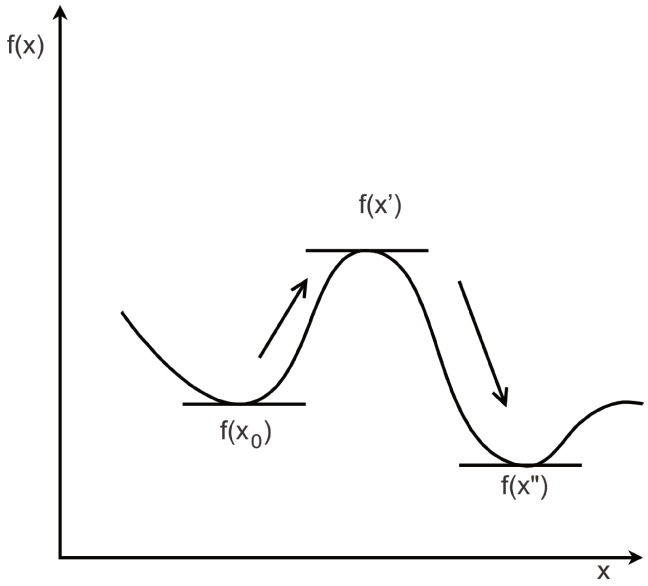


Figura 3.1: Ilustração do comportamento da função objetivo.

Observe que o algoritmo aceita um movimento de subida para que nas próximas soluções geradas seja possível encontrar uma solução melhor e assim escapar de locais sub-ótimos. Aqui, claramente, $f(x_0) > f(x'')$, e portanto, continua-se a gerar soluções na vizinhança $N(x'')$. É importante ressaltar que o *Simulated Annealing* foi desenvolvido para resolução de problemas de otimização combinatorial, e portanto, embora possa fornecer soluções não ótimas é possível alcançar soluções de boa qualidade em tempo computacional razoável.

A probabilidade de aceitação de uma solução pior que a atual é dada pelo *Algoritmo de Metropolis* [12] pela taxa:

$$P = exp\left(\frac{-\Delta}{T}\right),$$

onde: $\Delta = f_{viz} - f$, e T é um parâmetro de controle.

Frequentemente, na literatura, as etapas que descrevem o *Simulated Annealing* contêm características semelhantes, exceto pelo modo de gerar novas soluções em vizinhanças da solução corrente, em princípio, apresenta-se os seguintes passos:

1. Selecione uma temperatura inicial $T = T_0$, e uma solução inicial x_0. Defina $f_0 = f(x_0)$, $i = 0$, e siga para a Passo 2;

2. Faça $i = i + 1$;, gere aleatoriamente uma solução x_i e calcule $f_i = f(x_i)$;

3. Se $f_i < f_{i-1}$, vá para a Passo 5. Caso contrário, de acordo com a probabilidade $\| < exp\left(\frac{|f_i - f_{i-1}|}{T}\right)$, aceite x_i;

4. Se f_i foi rejeitada no passo 3, faça $f_i = f_{i-1}$;

5. Se estiver satisfeito com o valor da função objetivo f_i, então pare. Se não, decresça a temperatura T_0 é vá para o passo 2.

Com a finalidade de se intensificar mais a geração nas vizinhanças da solução corrente, opta-se pela versão do algoritmo que decresce a temperatura após L aceitações (por probabilidade) dentro da vizinhança. Desse modo, se o ponto gerado nos fornece um valor objetivo melhor do que o já conseguido, então abandona-se o *loop* interno e em seguida diminui-se a temperatura. Vejamos:

Dados

x	solução atual.
x'	ponto gerado na vizinhança.
$x^\star$	melhor solução até o momento.
T_0	temperatura inicial.
T_f	temperatura final.
L	nº máximo de iterações internas com aceitação.
$nmaxint$	nº máximo de iterações internas sem aceitação.

Ressaltamos a importância dos parâmetros nesse algoritmo, e para tal, será apresentado no próximo capítulo a seleção dos mesmos empregada no método.

Outra principal característica da heurística *simulated annealing* é a busca de soluções na vizinhança da solução incumbente, ou seja, a busca é feita a partir da melhor solução encontrada até o momento. A seguir, a próxima seção tem a finalidade de abordar a proposta

Algoritmo

[1] $x^\star = x$

[2] $T = T_0$

[3] $nmax = -L$

[4] WHILE $(T > T_f)$

$naceita = 0$
WHILE ($(naceita < L)$ && $(nmax < nmaint)$)
$nmax = namx + 1$
gerar$x' \in \mathcal{N}(x)$
$\Delta f = f(x') - f(x)$
IF $\Delta f < 0$
$x = x'$
$naceita = naceita + 1$
IF $f(x') < f(x^\star)$
$x^\star = x'$
$naceita = L$
END IF
ELSE
IF Uniforme$(0,1)$ $< exp(-\Delta f/T)$
$x = x'$
$naceita = naceita + 1$
END IF
END IF
END WHILE
$T = \alpha T$

[5] END WHILE

apresentada por *Chen* a qual desencadeou uma investigação para esse trabalho que, por sua vez, apresenta uma pesquisa mais abrangente em relação a busca local a partir de uma solução.

3.3 Geração de soluções

A busca por melhores soluções é feita a partir de uma perturbação na solução inicial factível, e segue através de pertubações da solução corrente. A proposta aqui apresentada é confrontada com a metodologia já existente em [2] publicado por *Chuen-Lung S. Chen , Stephen M. Hart, Wai Mui Tham.*

Uma vez que esse trabalho está fundamentado no artigo citado, será necessário primeiramente uma apresentação detalhada da busca local descrita por Chen, Hart e Tham, para depois apresentar nossa modificação.

3.3.1 Estratégia para Geração de Soluções Vizinhas - *Artigo*

Foi proposto anteriormente uma perturbação aleatória de apenas duas entradas da solução corrente, ou seja, dada uma solução propõe-se selecionar aleatoriamente duas entradas de modo que gere soluções vizinhas a partir da perturbação de um par de variáveis. Visto que são conhecidos os limites inferior e superior de cada variável de decisão, a estratégia propõe que para cada variável seja atribuído um valor entre seus respectivos limitantes, e além disso, a solução gerada deve ser factível.

Primeiramente, determina-se para cada variável os limitantes inferior e superior $[LB_j\,, UB_j]$. Claramente, $LB_j = 0, \quad \forall j = 1, \cdots, n$, uma vez que $x_j \geq 0, \quad \forall j = 1, \cdots, n$, e o limitante superior será o maior valor inteiro que pode ser atribuído a x_j para que seja possível atender uma demanda de d_i, $i = 1, \cdots, m$. Isto é, cada variável representa o número de vezes que um padrão de corte é utilizado, e em cada padrão pode ser cortado um item i

em uma quantidade denotada por a_{ij}, assim, o valor $\left\lceil \frac{d_i}{a_{ij}} \right\rceil$, $(a_{ij} \neq 0)$ é quantidade mínima que o padrão j deve ser cortado para atender a demanda d_i, portanto, define-se:

$$UB_j = \max_j \left\{ \left\lceil \frac{d_i}{a_{ij}} \right\rceil | i = 1, \cdots, m \;\; e \;\; a_{ij} \neq 0 \right\}, \quad \text{para cada} \;\; j = 1, \cdots, n.$$

Vejamos como proceder para determinar o limite inferior LB'_j para x'_j de modo que a solução x' seja factível.

Dada uma solução $x = [x_1 \cdots x_k \cdots x_n]^T$ factível, ou seja, $Ax = d + S$, com $S \geq 0$, tem-se:

$$\begin{bmatrix} a_{11}x_1 + & \cdots & +a_{1n}x_n \\ a_{21}x_1 + & \cdots & +a_{2n}x_n \\ \vdots & & \vdots \\ a_{m1}x_1 + & \cdots & +a_{mn}x_n \end{bmatrix} = \begin{bmatrix} d_1 & +S_1 \\ d_2 & +S_2 \\ \vdots & \vdots \\ d_m & +S_m \end{bmatrix}.$$

Ao selecionar uma variável x_k pode-se determinar seu limitante inferior LB'_k tal que a solução seja factível. De fato, para cada $i = 1, \cdots, m$, toma-se

$$u_i = \begin{cases} 0, & \text{se} \;\; a_{ik} = 0 \\ \left\lfloor \frac{S_i}{a_{ik}} \right\rfloor, & \text{se} \;\; a_{ik} > 0 \end{cases}.$$

A partir disso, denota-se $\Delta x_k = -\min_i \{u_i\}$, e define-se

$$LB'_k = \Delta x_k + x_k.$$

Observe que a solução x com $x'_k = \Delta x_k + x_k$ é factível pois para cada i tem-se:

$$\begin{aligned} a_{i1}x_1 + \cdots + a_{ik}x'_k + \cdots + a_{1n}x_n &= a_{i1}x_1 + \cdots - a_{ik}\left\lfloor \frac{S_l}{a_{lk}} \right\rfloor + a_{ik}x_k + \cdots + a_{1n}x_n \\ &= d_i + S_i - a_{ik}\left\lfloor \frac{S_l}{a_{lk}} \right\rfloor \geq d_i + S_i - a_{ik}\frac{S_i}{a_{ik}} \\ &= d_i \geq 0. \end{aligned} \tag{3.1}$$

Dada uma solução $x = (x_1, \cdots, x_n)$ em uma dada iteração do algoritmo, segue que para cada $j = 1, \cdots, n$ tem-se os limites inferior e superior LB_j e UB_j, e desse modo, inicia-se a busca na vizinhança de x, $N(x)$, descrita abaixo:

- Seleciona-se, aleatoriamente, duas variáveis de decisão (distintas), define-se $x_j = x'_j \in [LB_j, UB_j]$ e $x_k = x'_k \in [LB_k, UB_k]$;
- Se x', resultante, é factível, define-se $x = x'$ e $f = f(x')$, e segue-se com o SA;
- Caso contrário, define-se $x'_k = x_k$, e então determina-se $x_j = x'_j \in [LB'_j, UB_j]$ tal que x seja factível.

Entretanto, em exemplares com demandas por um maior número de peças distintas é possível gerar milhares de padrões de cortes, e cada um desses, por sua vez, é associado a uma variável de decisão, logo, após alguns teste, a geração de soluções melhores por meio da perturbação de apenas duas variáveis se mostra ineficiente, motivo pelo qual se desencadeou uma nova proposta para a estratégia de geração de soluções.

3.3.2 Uma Nova Estratégia para Geração de Soluções Vizinhas

Dada uma solução inicial, sabe-se que esta é factível (ou seja, $Ax \geq d$) e possui pelo menos m variáveis na base, além disso, possivelmente, não é a solução ótima e pode ser minimizada. Portanto, inicia-se uma perturbação com intenção de retirar e adicionar variáveis

na base atribuindo valores 0 ou 1, ou seja, com valor *zero* a variável sai da base, e com valor *um* a variável entra ou permanece na base, o fato mais importante é que a produção seja diminuída.

Estrategicamente, propõe-se uma pertubação na solução corrente de modo mais abrangente, ou seja, ponderadamente, a maior possível, e espera-se que seja possível obter uma solução que atenda a demanda mínima, isto é, a factibilidade é objetivada, entretanto, não é o objetivo inicial, visto que qualquer "alta produção" fornece uma solução factível, mas que não será a ótima. Portanto, direciona-se a solução corrente para uma solução infactível, isto é, "baixa produção", em seguida, o método aumenta a produção gradativamente até que se torne factível, e então segue com o algoritmo.

Outro diferencial incluído na estratégia de geração é o fato do cálculo do limitante de uma variável ser efetuado logo após a perturbação das variáveis, pois ao contrário de retornar o valor anterior de uma variável de decisão alterada, e consequentemente diminuir a variação de soluções geradas, torna-se mais coerente tentar atribuir o menor valor possível para que a solução seja factível.

A seguir são apresentadas as etapas que compõem a nova geração de soluções vizinhas:

(i) Encontra-se o conjunto de índices das variáveis de decisão não nulas.

(ii) Aleatoriamente, seleciona-se de 10% a 30% desses índices formando o conjunto I, e no máximo 3 índices dentre os índices das variáveis nulas, os quais comporão o conjunto K, então define-se, aleatoriamente $x_j = 0$ ou 1, com probabilidade *0.5*, para cada $j \in J$, onde $J = I \cup K$;

(iii) Se x', resultante, é factível, define-se $x = x'$,$f(x) = f(x')$ e segue-se com o SA;

(iv) Caso contrário, seleciona-se x'_j tal que $x'_j = \min_{k \in J}\{x'_k\}$, e então, determina-se LB'_j tal que x' seja factível;

- Se $LB'_j > UBj$, define-se $x'_j = x_j$, e testa-se a factibilidade novamente;

- Caso contrário, define-se $x'_j = LB'_j$ e $x = x'$, em seguida, segue-se com o SA.

No início do trabalho houvera algumas interrogações sobre a perturbação usada no artigo, basicamente sobre sua eficiência com problemas de porte maior. Portanto, tendo em mãos problemas maiores, foi possível fazer testes nos quais determinamos algumas características como número de variáveis a serem perturbadas, e de que modo essa perturbação seria mais eficiente. Desse modo, compomos o algoritmo anteriormente apresentado o qual se deu através de escolhas essenciais.

Capítulo 4

Experimentos computacionais

4.1 Introdução

Nossos experimentos computacionais estão embasados em exemplares gerados com características diferentes em relação a demanda e tamanho de itens. Após a obtenção de tais exemplares obtém-se soluções iniciais, soluções fornecidas pelos métodos *Chen* [2] e novo, e soluções obtidas pelo pacote computacional, *CPLEX*.

4.2 Classes de Problemas

Em 1995, Wäscher e Gau [6] constataram que em estudos de problemas de corte de estoque havia a necessidade de desenvolver algo mais sistemático, com capacidade de exercer a verificação de algoritmos melhores, no que se refere à soluções superiores, sobre outros algoritmos já publicados na literatura. A partir disso, na finalidade de tornar resultados mais satisfatórios, elaboraram uma proposta para desenvolver problemas-testes, a fim de disponibilizar uma verificação maior e menos tendenciosa. Para tanto, foi desenvolvido em [6] um gerador de problemas para Problemas de Corte de Estoque.

Para fins comparativos, usamos as 18 classes utilizadas por Salles-Neto [15], as quais foram geradas usando os mesmos parâmetros e a mesma semente, 1994, adotados no CUTGEN1 [6], e para cada classe selecionamos 20 exemplares. De forma sucinta, descreveremos aqui o modo como são gerados esses exemplares. A seguir, são dados os parâmetros:

Parâmetros

m Número de itens;

W Tamanho da bobina mestre;

v_1 Limite inferior dos comprimentos dos itens;

v_2 Limite superior dos comprimentos dos itens;

$\bar{d}$ Demanda média por item;

s Semente.

Os valores fixados para cada parâmetro estão descritos na Tabela 4.1

Parâmetros	Valores
m	10, 20, 40
W	1000
v_1	0.01 e 0.02
v_2	0.02 e 0.08
w_i	$v_1 \cdot W \leq w_i \leq v_2 \cdot W$
$\bar{d}$	10 e 100
s	1994

Tabela 4.1: Valores assumidos pelo CUTGEN1.

Desse modo, quando $w_i \in [0.01 \cdot W; 0.02 \cdot W]$ tem-se classes com itens pequenos para cada valor de m, e também, para cada valor de $\bar{d}$, assim, são formadas seis classes com itens pequenos. Analogamente, quando $w_i \in [0.01 \cdot W; 0.08 \cdot W]$ e, $w_i \in [0.02 \cdot W; 0.08 \cdot W]$, obtém-se

seis classes com itens variados e seis classes com itens grandes, respectivamente. Vejamos na Tabela 4.2 a classificação dos problemas gerados.

Classe	v_1	v_2	m	$\bar{d}$
Itens de tamanhos pequenos				
1	0.01	0.02	10	10
2	0.01	0.02	10	100
3	0.01	0.02	20	10
4	0.01	0.02	20	100
5	0.01	0.02	40	10
6	0.01	0.02	40	100
Itens de tamanhos variados				
7	0.01	0.08	10	10
8	0.01	0.08	10	100
9	0.01	0.08	20	10
10	0.01	0.08	20	100
11	0.01	0.08	40	10
12	0.01	0.08	40	100
Itens de tamanhos grandes				
13	0.02	0.08	10	10
14	0.02	0.08	10	100
15	0.02	0.08	20	10
16	0.02	0.08	20	100
17	0.02	0.08	40	10
18	0.02	0.08	40	100

Tabela 4.2: Características das classes geradas pelo CUTGEN1.

4.3 Seleção de Parâmetros

Após obter as classes com seus respectivos exemplares, o passo seguinte consiste em gerar as matrizes cujas colunas representam o padrões de corte.

4.3.1 Geração de padrões de corte

Como já mencionado no Capítulo 2, sabe-se que em determinados problemas é possível gerar milhares de padrões de corte, o que implica na inviabilidade de resolução exata em tempo razoável, portanto, em algumas classes, devido ao enorme número de padrões de corte, foram necessário impor um limite por tempo de trinta minutos na geração de colunas, e além disso, para que seja possível o uso do *solver CPLEX*, também é efetuada uma seleção dos padrões fornecidos, pois o *solver* atinge o limite de memória muito rapidamente. A seguir, apresenta-se na Tabela 4.3 a média de número de padrões de corte gerados em cada classe.

Classe	Número de Colunas		
	Menor	Média	Maior
1 & 2	6066	23434.85	55481
3 & 4	18468	41816.05	68644
5 & 6	23135	43763.35	54920
7 & 8	21	208.9	548
9 & 10	78	5450.3	29267
11 & 12	8449	23591.47	42475
13 & 14	12	34.25	65
15 & 16	27	142.15	289
17 & 18	400	966.6	2070

Tabela 4.3: Média entre os padrões gerados em cada classe.

4.3.2 Temperatura Inicial

Há vários meios de se estimar a temperatura T_0, por exemplo, reduzir a temperatura até que o número de soluções aceitas (por probabilidade) atinja 60%, e então usa-se a temperatura corrente como a inicial. Contudo, visto que a proposta é atacar todas as classes de modo geral, torna-se mais incisivo optar por testar em cada classe três níveis de temperatura e selecionar a melhor combinação entre eles. Portanto, tomamos um problema em cada classe,

executamos o método, e assim a Tabela 4.4 fornece os dados tais como valor objetivo e tempo de execução da seleção de T_0. Visto que a geração apresentada por *Chen* [2] é aplicada a apenas um exemplar e seus parâmetros são selecionados de acordo com o mesmo, aqui para diferentes classes, a seleção será determinada de acordo com testes da metodologia própria, isto se deve ao fato de que alguns testes feitos com a metodologia do artigo não obtiveram qualquer resultado significativo.

Classe	$T_0 = 8000$		$T_0 = 4000$		$T_0 = 2000$	
	Valor Obj.	Tempo(s)	Valor Obj.	Tempo(s)	Valor Obj.	Tempo(s)
1	10419	448.9	10220	443.7	**9221**	450.6
2	27136	549.3	**20339**	608.5	20935	609.0
3	107996	1014.7	83045	1079.8	**78380**	1368.4
4	549554	1529.6	**543803**	1483.9	547553	1499.9
5	339633	6360.3	325385	8447.1	**310882**	8497.9
6	2943365	2812.6	2940695	2583.1	**2935695**	2564.2
7	5050	0.2	3051	0.22	**3050**	0.21
8	4401	0.18	**1394**	0.22	2393	0.2
9	**70093**	261.5	70493	284	70497	306.9
10	283937	215	**280436**	221.2	280936	217.9
11	44202	264.8	42201	286.6	**38198**	305.3
12	389667	385.9	384665	362.77	**383665**	369.38
13	5855	0.13	3053	0.13	**1853**	0.13
14	3529	0.1	**2330**	0.11	2730	0.12
15	19127	0.72	18126	0.73	**16928**	0.76
16	7033	0.72	**2033**	0.33	3037	0.73
17	28252	7.49	27247	8.2	**25247**	8.93
18	214144	3.82	**211151**	3.72	212148	3.84

Tabela 4.4: Seleção de Temperatura Inicial.

Portanto, é possível notar que uma temperatura alta permite a aceitação de muitas soluções que não são de melhora o que talvez faça a busca se *distanciar* da solução ótima, e dessa forma, torna-se menos eficiente. Sendo assim, opta-se por uma temperatura menor, e devido aos resultados dos testes, a temperatura inicial foi definida como entre as temperatura iniciais que forneceram melhores resultados, sendo assim, a temperatura inicial fica

determinada por $T_0 = 3000$.

4.3.3 Números de Iterações Internas

Assim como na seleção de temperatura inicial aplicamos somente a metodologia própria, pois na metodologia apresentada por *Chen* [2], fixamos o número de iterações internas como $L = 16 \cdot n$, igualmente como foi especificado pelo autor. A seleção aqui apresentada visa encontrar o parâmetro que se encaixa melhor em relação ao valor objetivo e tempo de máquina, portanto, tendo observado que o parâmetro tem relação direta com o número de variáveis, é suficiente selecionar um exemplar que contenha um número expressivo de variáveis e a partir deste verificar por testes qual valor deve ser atribuído ao parâmetro L, e, para a metodologia proposta, a Tabela 4.5 reporta os dados dos testes efetuados em um exemplar da Classe 5.

Valor L	Valor Objetivo	Tempo (s)
$n = 52971$	339140	1390,5
$2 \cdot n$	335143	2981,6
$4 \cdot n$	332141	5515,7
$8 \cdot n$	325139	11183
$16 \cdot n$	319138	22145

Tabela 4.5: Seleção do número de iterações internas.

Facilmente observa-se que para cada valor maior de números de iterações internas é possível encontrar soluções melhores. Contudo, objetiva-se também por tempo razoável de resoluções e portanto tem-se como razoável o valor fixado em $L = 4 \cdot n$

4.4 Análise de Resultados

A análise de resultados está dividida na comparação entre as metodologias pesquisadas, e em princípio analisa-se resultados obtidos pelo algoritmo do artigo estudado e o algoritmo

apresentado nesse trabalho, na sequência, compara-se resultados obtidos pela metodologia própria com resultados fornecidos pelo *solver* comercial *CPLEX*.

De acordo com a formulação matemática do problema aqui proposta, em cada problema será considerado os custos C_1 e C_2, e uma penalidade $M = 1000$ por unidade excedente. Após a implementação e execução dos algoritmos é possível extrair das soluções encontradas informações tais como números de itens excedentes, *setup*, itens processados, e desperdício provenientes do padrões selecionados, e portanto, os dados extraídos terão suas médias relatadas em tabelas.

Nas tabelas apresentadas nas análises a seguir estará incluída uma terna denotada por (m, t, d) em que m representa o número de itens, t denota a variação dos tamanhos dos itens e d a demanda. Vejamos, rapidamente, a descrição aqui definida dada na Tabela 4.6.

Número de Itens (m)	Tamanho (t)	Demanda (d)
10	P - Pequeno	B - Baixa
20	V - Variados	A - Alta
40	G - Grande	-

Tabela 4.6: Notação denominada para cada classe.

Todos os algoritmos foram implementados e executados na linguagem do programa *MatLab* (MATLAB R2010b 64-bit), além disso, o pacote comercial *CPLEX* (64-bit) foi usado como extensão no *MatLab*. Os programas foram rodados em dois microcomputadores, sendo que o *Simulated Annealing* foi totalmente executado no microcomputador *Intel Core 2 (2.33 GHz), 4 GB (RAM) e sistema operacional Windows 7 (64 Bits)*, no entanto, devido a um problema de incompatibilidade ocorrido entre o *MatLab* e o *solver*, a alternativa foi executar o *CPLEX* no microcomputador com processador *Intel Core 2 Duo (1.66 GHz), 2 GB (RAM) e sistema operacional Windows 7 (64 Bits)*.

4.4.1 Comparação de Soluções Geradas para o SA : *Nova vs Artigo*

Considerando $C_1 = C_2 = 1$, a execução nos vinte exemplares em cada classe fornece as Tabelas 4.7, 4.8, e 4.9 que representam as classes de itens de tamanhos pequenos, itens de tamanhos variados e itens de tamanhos grandes, respectivamente, que por sua vez apresentam as médias do número de *setup*, objetos processados, itens excedentes à produção e desperdícios provenientes em cada padrão de corte selecionado, no entanto, esse último não é considerado na formulação da função objetivo e portanto não deve ser avaliado de modo comparativo, mas como informação adicional. Contudo, mais adiante apresentaremos uma tabela onde são considerados os custos totais e, em particular, os desperdícios por excesso de produção e os obtidos ao final de cada padrão de corte.

Observando os dados relatados na Tabela 4.7 é possível notar a superioridade dos resultados da nova geração de soluções. Isto se deve ao fato de que, em muitos exemplares, a geração de soluções publicada no artigo ser incapaz de gerar uma solução melhor do que a dada como solução inicial. Para as Classes 2, 4 e 6, vê-se que a média do número de *setup* é maior nas soluções geradas pela nova proposta, entretanto, o número de objetos processados e o excesso de produção são menores em todas as Classes (1 até 6).

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
	(m, t, d)	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo
1	(10, P, B)	**7.2**	9.85	**11.4**	15.95	**5.6**	80.2	5	**0.2**
2	(10, P, A)	12.2	**9.8**	**108.8**	112.65	**6.5**	65.2	31.45	**0.15**
3	(20, P, B)	**14.45**	17.6	**34.5**	41.65	**67.7**	149.4	6.45	**0**
4	(20, P, A)	19.2	**17.65**	**325.4**	331.25	**528.95**	589.05	3.45	**0**
5	(40, P, B)	**30.45**	33.15	**122.15**	132.45	**427.1**	508.7	14.45	**9.6**
6	(40, P, A)	34.1	**33.6**	**1195.5**	1204.9	**4065.3**	4136.5	11.6	**10.2**

Tabela 4.7: Resultados com Estratégias: Proposta e Artigo - $C_1 = C_2 = 1$ (Classes: 1-6).

Para as classes com itens de tamanhos variados apresentados na Tabela 4.8, constata-se, novamente, que para algumas classes, especificamente Classes 8 e 10, o número médio de

setup é menor nas soluções encontradas pela estratégia de geração do artigo, contudo, as médias de número de objetos processados e de excesso de produção são maiores nas Classes 7 até 12, ou seja, é possível afirmar que *setup* e número objetos processados são inversamente proporcionais.

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
(m,t,d)		(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo
7	(10,V,B)	**7.95**	8.1	**47.4**	48.25	**51**	55.1	244.55	**206.85**
8	(10,V,A)	9	**8.2**	**471.1**	472	**504.7**	509.4	246.45	**209.4**
9	(20,V,B)	**15.7**	16.4	**99.35**	101.5	**64.85**	70.9	**418.55**	425.6
10	(20,V,A)	16.7	**16.65**	**988.7**	991.15	**635.55**	642.55	431.7	**430.35**
11	(40,V,B)	**28.35**	30.25	**181.6**	185.75	**152.2**	165.7	**167.45**	175.35
12	(40,V,A)	**30.9**	31.3	**1777.1**	1781.6	**1311.9**	1326.7	**188.4**	190.3

Tabela 4.8: Resultados com geração de soluções: proposta e artigo - (Classes: 7-12).

Novamente, na Tabela 4.9 as médias de número de *setup* nas Classes 13, 14 e 18 também são menores nas soluções encontradas pela geração especificada no artigo, no entanto, assim como nas tabelas já mencionadas, a geração de soluções proposta nesse trabalho gera soluções com menor número de objetos processados e consequentemente com menor número de excesso de produção.

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
(m,t,d)		(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo
13	(10,G,B)	9.2	**9**	**62.2**	62.8	**20.8**	23.05	812.9	**764.85**
14	(10,G,A)	9.45	**9**	**620.35**	620.8	**207.45**	208.9	814.55	**764.8**
15	(20,G,B)	**17.1**	17.6	**125.5**	125.8	**39.7**	41.3	**1132.5**	1215.1
16	(20,G,A)	**18.1**	18.2	**1248.3**	1248.6	**388.2**	389.1	**1154.3**	1245
17	(40,G,B)	**34.4**	35.4	**216.4**	217.2	**31.3**	33.4	**1519.6**	1630.8
18	(40,G,A)	37.1	**36**	**2191.5**	2192.7	**362.1**	365.4	1655.7	**1627.4**

Tabela 4.9: Resultados com geração de soluções: proposta e artigo - (Classes: 13-18).

É importante ressaltar que em inúmeros exemplares de cada uma das classes abordadas as soluções geradas pela estratégia especificada no artigo não superam as soluções

iniciais, e devido a esse fato nota-se que o número médio de *setup* pode ser menor em algumas classes, uma vez que a solução inicial é fornecida pela resolução do problema relaxado sem as restrições de *setup*.

4.4.2 Comparação entre *Simulated Annealing* e *CPLEX*

Nesse momento, buscamos uma comparação com o *solver* comercial, *CPLEX*, de modo que seja feita uma avaliação da qualidade dos resultados obtidos pela heurística composta com a nova proposta de geração de soluções. Ressalta-se que apesar da necessidade de se limitar o número de colunas em várias classes para que o *solver CPLEX* consiga nos fornecer uma solução ótima inteira, este ainda se mostra superior na maioria dos exemplares. Dependendo da classe, o *CPLEX* reporta uma solução exata que muitas vezes é uma solução aceita por tolerância (i. e., o *CPLEX* para quando há uma diferença mínima entre a solução atual e a possível solução ótima, e então reporta a atual como solução ótima), e em muitas classes o programa é parado por falta de memória, nesse caso o usuário interrompe e armazena a solução encontrada até o instante. As Tabelas 4.10, 4.11 e 4.12 apresentam as médias do número de *setup*, número de objetos processados e excesso de produção, fornecidas através dos resultados obtidos. Novamente, uma vez que o desperdício não é considerado diretamente na formulação ressalta-se que é apresentada a sua média apenas como informação adicional.

Analisando as Classes 1 até 6, em que os itens têm tamanhos pequenos, é possível observar, nas Classes 4, 5 e 6, que o número médio de *setup* é menor nas soluções encontradas pelo *Simulated Annealing*, entretanto, nota-se que o *solver* fornece soluções superiores no que se refere a média de objetos processados e excesso de produção.

É razoável concluir o fato de que um número maior de *setup* é refletido em uma diminuição no número de objetos processados, que por sua vez, gera menos itens excedentes à demanda. Portanto, apesar do fato do número médio de *setup* ser menor nas soluções encontradas pelo SA nas Classes 1,2 e 3, é visível o aumento no processamento de obje-

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
	(m,t,d)	SA (N)	CPLEX	SA (N)	CPLEX	SA (N)	CPLEX	SA (N)	CPLEX
1	(10, P, B)	7.2	**5.75**	11.4	**11.3**	5.6	**4.45**	5	**3.55**
2	(10, P, A)	12.2	**11.9**	108.8	**108.6**	6.5	**2.5**	31.45	**19.7**
3	(20, P, B)	14.45	**14.15**	34.5	**32.9**	67.7	**54.6**	6.45	**3.3**
4	(20, P, A)	**19.2**	19.25	325.4	**323.65**	528.95	**511.7**	3.45	**3.05**
5	(40, P, B)	**30.45**	31.90	122.15	**120**	427.1	**408.2**	**14.45**	15.95
6	(40, P, A)	**34.1**	36.5	1195.5	**1193**.3	4065.3	**4045.7**	**11.6**	14.4

Tabela 4.10: Resultados com $C_1 = C_2 = 1$ (Classes: 1-6).

tos do estoque e consequentemente um maior excesso de produção, segundo as informações apresentadas na Tabela 4.10.

Os valores médios apresentados nas Classes 7 até 12 (Tabela 4.11) apresentam uma pequena diferença entre os métodos, e em uma análise de um ponto de vista mais geral, no que se refere à variação de custos totais, será possível observar esse fato com mais clareza.

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
	(m,t,d)	SA (N)	CPLEX	SA (N)	CPLEX	SA (N)	CPLEX	SA (N)	CPLEX
7	(10,V,B)	7.95	**7.1**	244.55	**217.9**	51	**50.75**	47.40	**47.25**
8	(10,V,A)	9	**8.2**	**471.1**	**471.1**	504.70	**504.35**	246.45	**220.75**
9	(20,V,B)	15.7	**14.2**	**99.35**	**99.35**	64.85	**64**	418.55	**417.3**
10	(20,V,A)	16.7	**15.85**	**988.7**	989.45	635.55	**635.05**	431.7	**425.55**
11	(40,V,B)	28.35	**26.25**	181.6	**180.3**	152.2	**148.45**	167.45	**159.9**
12	(40,V,A)	30.9	**30.6**	1777.1	**1776.1**	1311.9	**1309.5**	188.4	**179.2**

Tabela 4.11: Resultados com $C_1 = C_2 = 1$ (Classes: 7-12).

A Tabela 4.12 refere-se a resultados obtidos em classes com demanda por itens grandes. Novamente, o solver fornece soluções superiores em relação as médias de número de *setup* e excesso de produção, mas visivelmente com pouca diferença na maioria dos valores comparados. Para que haja mais clareza na comparação uma tabela com a variação da diferença em porcentagem será apresentada mais adiante.

Os resultados apresentados até agora foram referentes aos problemas onde custos de

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
	(m,t,d)	SA (N)	CPLEX	SA (N)	CPLEX	SA (N)	CPLEX	SA (N)	CPLEX
13	(10,G,B)	9.2	**8**	62.2	**62.15**	20.8	**20.75**	812.9	**682.95**
14	(10,G,A)	9.45	**8.85**	**620.35**	620.60	207.45	**207.25**	814.55	**755.05**
15	(20,G,B)	17.1	**15**	125.5	**125.2**	39.7	**39**	1132.5	**1096.7**
16	(20,G,A)	18.1	**17.6**	**1248.3**	**1248.3**	388.2	**387.9**	**1154.3**	1270.8
17	(40,G,B)	34,4	**29,9**	216,4	**215,9**	31,3	**29,2**	**1519.6**	1612.3
18	(40,G,A)	37.1	**34.2**	2191.5	**2191**	362.1	**360.3**	1655.7	**1583.9**

Tabela 4.12: Resultados com $C_1 = C_2 = 1$ (Classes: 13-18).

setup e custos por Objetos Processados são iguais, todavia, questiona-se quais mudanças ocorrem se atribuirmos uma valor maior para cada *setup* na função objetivo, e portanto, na próxima seção efetuamos essa verificação.

4.5 Análise com Aumento no Custo Fixo de *Setup*

Intencionalmente, atribui-se um custo fixo de *setup*, $C_2 = 10$, na formulação, e aplica-se novamente a heurística com ambas estratégias de geração de soluções vizinhas e o *solver CPLEX* sobre todos os exemplares gerados em cada classe. A partir disso, uma nova análise é feita com base nos experimentos computacionais com a intenção de investigar o comportamento dos resultados obtidos. A seguir são apresentadas as Tabelas 4.13, 4.14 e 4.15, que fornecem dados comparativos entre as duas estratégias de geração de soluções vizinhas, e, para efeito comparativo da qualidade de soluções, apresenta-se as Tabelas 4.16, 4.17 e 4.18, em que são fornecidas as informações obtidas das soluções encontradas pela heurística e pelo *solver*. Em todas são informados os valores médios dos números de *setup*, objetos processados e itens excedentes à demanda, além disso, como informação adicional tem-se a média de desperdícios obtidos nos padrões de corte selecionados pelas soluções encontradas.

4.5.1 Comparação entre as estratégias: *(Nova vs Artigo)*

Visivelmente, a metodologia da geração de soluções apresentada em *Chen* [2] mostra pouquíssimas variações. Na maioria das classes pode-se observar uma pequena diferença ao que foi apresentado nas Tabelas 4.7, 4.8 e 4.9, porém em nenhuma houve uma melhora significativa. Imediatamente, conclui-se que esse fato ocorre, pois, na maioria dos exemplares de cada classe, o método implementado por *Chen* [2] é incapaz de gerar soluções melhores do que as iniciais.

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
	(m,t,d)	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo
1	(10, P, B)	**7.15**	9.8	**11.95**	15.95	**9.6**	80.2	4.85	**0.2**
2	(10, P, A)	12.9	**9.8**	**109.1**	112.65	**9.45**	65.2	42.8	**0.45**
3	(20, P, B)	**14.75**	17.6	**34.35**	41.65	**66.35**	149.4	6.7	**0**
4	(20, P, A)	18.7	**17.65**	**325.35**	331.25	**528.4**	589.05	2.95	**0**
5	(40, P, B)	**30.25**	33.05	**121.35**	132.45	**420.25**	508.8	15.05	**9.65**
6	(40, P, A)	34.2	**33.7**	**1195.5**	1204.8	**4064.6**	4135.9	11.8	**10.1**

Tabela 4.13: Resultados das médias com $C_1 = 1$; $C_2 = 10$

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
	(m,t,d)	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo
7	(10,V,B)	**8.1**	8.35	**47.75**	48.3	**52.65**	55.25	222.85	**222.5**
8	(10,V,A)	8.65	**8.2**	**471.45**	471.9	**506.35**	509.2	241.45	**209.4**
9	(20,V,B)	**15.65**	16.45	**99.5**	101.65	**65.35**	71.2	**412.3**	426.05
10	(20,V,A)	**16.5**	16.65	**988.75**	991.25	**635.9**	642.75	**423.95**	430.15
11	(40,V,B)	**28.15**	30.25	**181.6**	185.7	**152.45**	165.65	**161.55**	175.35
12	(40,V,A)	**30.6**	31.1	**1801.3**	1805.8	**1475.7**	1489.6	**178.1**	182.1

Tabela 4.14: Resultados das médias com $C_1 = 1$; $C_2 = 10$ - (Classes: 7-12).

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
(m,t,d)		(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo	(Nova)	Artigo
13	(10,G,B)	9.45	**9**	**62.15**	62.85	**20.85**	23.15	833.8	**764.85**
14	(10,G,A)	10.4	**9**	**620.4**	620.8	**207.3**	208.9	923.4	**764.8**
15	(20,G,B)	17.7	**17.6**	**125.3**	125.9	**39.3**	41.4	**1167.3**	1215.1
16	(20,G,A)	18.6	**18.1**	**1248.3**	1248.7	**388.1**	389.2	1243.2	**1241.4**
17	(40,G,B)	**34.9**	35.4	**216.4**	217.2	**31.1**	33.4	**1597.3**	1630.8
18	(40,G,A)	**36**	**36**	**2191.1**	2192.8	**361.5**	365.6	**1541.5**	1627.4

Tabela 4.15: Resultados das médias com $C_1 = 1$; $C_2 = 10$ - (Classes: 13-18).

4.5.2 Comparação entre *Simulated Annealing* e *CPLEX*

Para as classes de itens com tamanhos pequenos, onde as informações apresentadas na Tabela 4.16 são descritas, é possível observar claramente que o número médio de *setup* é menor nas soluções encontradas pelo *Simulated Annealing* nas Classes 2 até 6, entretanto, como consequência tem-se uma diferença relativamente grande entre os valores médios de objetos processados, e por essa razão uma acentuada diferença entre excesso médio de produção, principalmente nas Classes 1, 2 e 3.

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
(m,t,d)		SA (N)	*CPLEX*	SA (N)	*CPLEX*	SA (N)	*CPLEX*	SA (N)	*CPLEX*
1	(10, P, B)	7.15	**6.55**	11.95	**11.35**	9.6	**5.1**	**4.85**	5.65
2	(10, P, A)	**12.9**	13.8	109.1	**108.75**	9.45	**3.25**	42.8	**31.6**
3	(20, P, B)	**14.75**	**14.75**	34.35	**33.2**	66.35	**57.4**	6.7	**2.6**
4	(20, P, A)	**18.7**	21.65	325.35	**323.8**	528.4	**514.4**	**2.95**	4
5	(40, P, B)	**30.25**	31.05	121.35	**120.3**	420.25	**410.8**	15.05	**13.1**
6	(40, P, A)	**34.2**	35.0	1195.5	**1193.7**	4064.6	**4048.6**	**11.8**	12.6

Tabela 4.16: Resultados com $C_1 = 1$; $C_2 = 10$ (Classes: 1-6).

A Tabela 4.17 expõe as informações dadas pelas soluções obtidas nas classes com itens de tamanhos variados. Observe que as médias apresentadas demonstram poucas diferenças entre a aplicação das metodologias, e isso implica que em uma comparação de custos totais existe uma diferença relativamente pequena entre ambas.

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
(m,t,d)		SA (N)	*CPLEX*	SA (N)	*CPLEX*	SA (N)	*CPLEX*	SA (N)	*CPLEX*
7	(10,V,B)	8.1	**7**	47.75	**47.4**	52.65	**50.75**	**222.85**	232.5
8	(10,V,A)	8.65	**7.85**	**471.45**	471.5	506.35	**504.35**	241.45	**218.25**
9	(20,V,B)	15.65	**13.95**	99.5	**99.35**	65.35	**64**	**412.3**	415.6
10	(20,V,A)	16.5	**15.8**	**988.75**	990.35	635.9	**635.05**	**423.95**	439.7
11	(40,V,B)	28.15	**27.1**	181.6	**180.7**	152.45	**148.45**	**161.55**	202.9
12	(40,V,A)	30.6	**30**	1801.3	**1800.9**	1475.7	**1472.7**	**178.1**	187.1

Tabela 4.17: Resultados com $C_1 = 1$; $C_2 = 10$ (Classes: 7-12).

Embora o número médio de *setup* seja menor nas soluções dadas pelo *solver* em todas as classes da Tabela 4.18, vê-se que, consequentemente, o número médio de objetos processados é maior do que os das soluções obtidas pelo *Simulated Annealing* (exceto na Classe 17), entretanto, ainda que uma diferença pequena, o número médio de itens excedentes à demanda são menores nas soluções encontradas pelo *solver*.

Classificação		*Setup*		Obj. Processados		Excesso de produção		Desperdício	
(m,t,d)		SA (N)	*CPLEX*	SA (N)	*CPLEX*	SA (N)	*CPLEX*	SA (N)	*CPLEX*
13	(10,G,B)	9.45	**7.7**	**62.15**	62.55	20.85	**20.75**	833.8	**703.35**
14	(10,G,A)	10.4	**8.3**	**620.4**	621.5	207.3	**207.25**	923.4	**721**
15	(20,G,B)	17.7	**14.7**	**125.3**	125.6	39.3	**39**	1167.3	**1101.3**
16	(20,G,A)	18.6	**15.7**	**1248.3**	1248.9	388.1	**387.9**	1243.2	**1121.8**
17	(40,G,B)	34.9	**29.9**	216.4	**215.9**	31.1	**29.2**	**1597.3**	1612.3
18	(40,G,A)	36	**34.3**	**2191.1**	2192.2	361.5	**360.3**	**1541.5**	1709.3

Tabela 4.18: Resultados com $C_1 = 1$; $C_2 = 10$ (Classes: 13-18).

Observe que embora o custo por *setup* tenha um aumento, ainda tem-se o excesso de produção fortemente penalizado na função objetivo, logo, é possível que as soluções encontradas admitam um maior número de *setup* pois o custo por item excedente continua sendo alto.

4.5.3 Análise de Custos Totais

Ao se considerar os itens excedentes também como desperdícios façamos a soma dos produtos entre a $sobra_i$ e o tamanho w_i para cada $i = 1, \cdots, m$, ou seja:

$$\sum_i sobra_i \cdot w_i\,, \qquad i = 1, \cdots, m.$$

A seguir, como informações adicionais, são apresentados nas Tabelas 4.19, 4.20, o número médio de desperdício referente aos itens excedentes à demanda.

Classificação		Desperdício Médio		
	(m, t, d)	SA (Art.)	SA (Novo)	*CPLEX*
1	(10,P,B)	5179.2	623.9	**521.4**
2	(10,P,A)	4605.5	681.5	**331.2**
3	(20,P,B)	19070.8	11913.2	**10314.3**
4	(20,P,A)	105651.4	99797.2	**97992.9**
5	(40,P,B)	90667.2	80363.2	**78194.9**
6	(40,P,A)	787201.4	777810.5	**775490.3**
7	(10,V,B)	5797.6	4848.8	**4728.5**
8	(10,V,A)	46859.3	45940.3	**45831.7**
9	(20,V,B)	11393.8	9296.2	**9085.7**
10	(20,V,A)	91805	89596	**89542.6**
11	(40,V,B)	23971	19855.2	**19464.5**
12	(40,V,A)	176977.8	172424.6	**171440.9**
13	(10,G,B)	6055.7	5797.7	**5784.7**
14	(10,G,A)	56762.7	56257.5	**56025.8**
15	(20,G,B)	11879.4	11737.8	**11000.8**
16	(20,G,A)	113257.9	112950.5	**108720.4**
17	(40,G,B)	9233.1	9007.3	**8407.4**
18	(40,G,A)	105924.1	104556.2	**98546.8**

Tabela 4.19: Desperdício por Excesso de Produção - $C_1 = C_2 = 1$.

Diferentemente do que é considerado diretamente na função objetivo, os custos totais representam as somas de número de objetos processados, *setup*, desperdício relativo ao final de cada padrão de corte juntamente com a soma dos tamanhos de itens finais excedentes à

Classificação		Desperdício Médio		
	(m,t,d)	SA (Art.)	SA (Novo)	*CPLEX*
1	(10,P,B)	5179.1	1173.1	**567.6**
2	(10,P,A)	4605.2	978.5	**428.1**
3	(20,P,B)	19070.8	11762.6	**10616.2**
4	(20,P,A)	105651.4	99748	**98166.2**
5	(40,P,B)	90666.8	79562.4	**78503.7**
6	(40,P,A)	787102.8	777860.6	**775876.4**
7	(10,V,B)	5840.3	5262.8	**4958.2**
8	(10,V,A)	46766.6	46279.4	**45827.7**
9	(20,V,B)	11491.4	9414.8	**9064.9**
10	(20,V,A)	91898.6	89670.7	**88843.9**
11	(40,V,B)	23922.5	19854.4	**19706.2**
12	(40,V,A)	188833.3	184407.8	**183712**
13	(10,G,B)	6101.9	**5534.6**	5771.6
14	(10,G,A)	58762.7	**53918.7**	57977.2
15	(20,G,B)	12426	11768.1	**11354.3**
16	(20,G,A)	111305.2	**110094.5**	110383
17	(40,G,B)	9233.1	8731.4	**8207.4**
18	(40,G,A)	105023.1	104461.3	**98769.6**

Tabela 4.20: Desperdício por Excesso de Produção - $C_1 = C_2 = 10$.

produção, isto é:

$$\text{Custo Total} = C_1 \sum_j x_j + C_2 \sum_j y_j + \sum_i sobra_i \cdot w_i + \sum_j desp_j \cdot x_j.$$

Como já mencionado, na maioria dos experimentos o *CPLEX* se mostra superior em grande parte dos dados obtidos a partir das soluções encontradas, contudo, especifica-se essa diferença, entre a heurística com a nova proposta de geração de soluções vizinhas e o *solver*, nas Tabelas 4.21 e 4.22 onde são dados os valores médios dos custos totais obtidos em cada classe, e além disso, a variação da diferença em porcentagem entre cada método.

Classificação		Custos Totais		Variação da
	(m,t,d)	SA (N)	*CPLEX*	Diferença em %
1	(10,P,B)	647.5	**542**	16.29
2	(10,P,A)	833.9	**471.4**	43.47
3	(20,P,B)	11968.6	**10364.6**	13.40
4	(20,P,A)	100145.2	**98338.8**	1.80
5	(40,P,B)	80530.2	**78362.7**	2.69
6	(40,P,A)	779051.7	**776734.5**	0.29
7	(10,V,B)	5148.7	**5000.7**	2.87
8	(10,V,A)	46666.8	**46531.7**	0.28
9	(20,V,B)	9829.8	**9616.5**	2.16
10	(20,V,A)	91033.1	**90973.4**	0.06
11	(40,V,B)	20232.6	**19830.9**	1.98
12	(40,V,A)	174421	**173426.8**	0.57
13	(10,G,B)	6682	**6537.8**	2.15
14	(10,G,A)	57701.8	**57410.3**	0.50
15	(20,G,B)	13012.9	**12237.5**	5.95
16	(20,G,A)	115371.2	**111257.1**	3.56
17	(40,G,B)	10777.7	**10265.5**	4.75
18	(40,G,A)	108440.6	**102355.9**	5.61

Tabela 4.21: Variação de Custos Totais $C_1 = C_2 = 1$.

Embora em alguns exemplares das Classes 1, 2 e 3 seja possível encontrar uma solução melhor do que a fornecida pelo *CPLEX*, é notável a diferença acentuada entre as metodologias nos demais exemplares. Por outro lado, quando atribui-se um custo maior para cada *setup*,

isto é $C_2 = 10$, é possível notar uma melhora nas Classes 13, 14 e 16 (cf. Tabela 4.20).

Classificação		Custos Totais		Variação da
	(m, t, d)	SA (N)	*CPLEX*	Diferença em %
1	(10,P,B)	1261.4	**650.1**	48.46
2	(10,P,A)	1259.4	**706.4**	43.90
3	(20,P,B)	11951.1	**10799.5**	9.63
4	(20,P,A)	100263.3	**98710.5**	1.54
5	(40,P,B)	80001.3	**78947.6**	1.31
6	(40,P,A)	779409.9	**777432.7**	0.25
7	(10,V,B)	5614.4	**5308.1**	5.45
8	(10,V,A)	47078.9	**46595.9**	1.02
9	(20,V,B)	10083.1	**9719.3**	3.60
10	(20,V,A)	91248.4	**90431.9**	0.89
11	(40,V,B)	20479	**20360.8**	0.57
12	(40,V,A)	186693.2	**186000**	0.37
13	(10,G,B)	**6525**	6614.5	-1.37
14	(10,G,A)	**55566.5**	59402.7	-6.90
15	(20,G,B)	13237.7	**12728.2**	3.84
16	(20,G,A)	**112772**	112910.7	-0.12
17	(40,G,B)	10894.1	**10334.6**	5.13
18	(40,G,A)	108553.9	**103014.1**	5.10

Tabela 4.22: Variação de Custos Totais $C_1 = 1$; $C_2 = 10$.

Por fim, apresenta-se na Tabela 4.23 o valor médio do tempo de execução de cada metodologia, entretanto, ressalta-se que os tempos em relação a execução do *Simulated Annealing* não devem ter efeito comparativo em relação ao tempo decorrido pelo *solver CPLEX*, uma vez que não foram implementados na mesma linguagem nem no mesmo computador, ou seja, são apenas dados informativos.

Classificação		Tempo(s) $C_1 = 1$ & $C_2 = 1$			Tempo(s) $C_1 = 1$ & $C_2 = 10$		
	(m, t, d)	SA (Art.)	SA (N)	*CPLEX*	SA (Art.)	SA (N)	*CPLEX*
1	(10,P,B)	**206.18**	304.29	300.02	**185.3**	285.9	301.1
2	(10,P,A)	379.43	512.44	**320.9**	271.3	425.5	**181.6**
3	(20,P,B)	1870.9	1840	**302.66**	1827.1	1959.8	**130.8**
4	(20,P,A)	2187.1	1792.7	**407.1**	1409.3	1437.3	**172.3**
5	(40,P,B)	5338.2	3106.5	**249.6**	4515.2	2832.4	**183.7**
6	(40,P,A)	6127.7	3147.6	**318.3**	6085.8	3132.7	**196.3**
7	(10,V,B)	0.33	**0.2**	46,23	0.37	**0.24**	35.7
8	(10,V,A)	0.36	**0.22**	52.1	0.41	**0.25**	88.9
9	(20,V,B)	122.9	99.8	**94.7**	123.84	99.7	**90.4**
10	(20,V,A)	123.1	104.9	**71.1**	75.1	69.2	**53.9**
11	(40,V,B)	2078.8	1126.1	**230.3**	1319.1	783.7	**186.3**
12	(40,V,A)	2162.7	1150.4	**206.2**	1333.7	772.6	**181.3**
13	(10,G,B)	0.09	**0.05**	0.23	0.10	**0.05**	0.31
14	(10,G,A)	0.09	**0.04**	3.3	0.10	**0.06**	9.4
15	(20,G,B)	0.45	**0.16**	11.9	0.45	**0.18**	39.7
16	(20,G,A)	0.48	**0.19**	14.58	0.49	**0.21**	34.7
17	(40,G,B)	7.1	**1.7**	100.32	6.5	**1.5**	100.3
18	(40,G,A)	7.2	**1.8**	83.38	7.4	**1.9**	129.3

Tabela 4.23: Tempos decorridos pelo método de resolução.

Capítulo 5

Considerações Finais & Perspectivas

A partir dos resultados fornecidos pelos experimentos computacionais é possível concluir que a nova proposta de gerar soluções fornece soluções com qualidade superior em todas as classes geradas quando comparada com a estratégia utilizada por Chen [2].Portanto, uma vez que uma das motivações dessa investigação era gerar vizinhanças melhores, pois, a geração apresentada em Chen [2] é limitada , de fato, consegue-se comprovar uma superioridade sobre a geração publicada em [2].

É importante destacar que nos resultados fornecidos pela nova estratégia obteve-se em todos os exemplares de cada uma das classes uma solução melhor do que a dada como solução inicial, enquanto que em muitos exemplares a estratégia de Chen [2] não conseguiu encontrar uma solução melhor do que a dada como inicial.

Embora em algumas classes o *Simulated Annealing* consiga soluções com o valor médio de *setup* menor, isso não ocorreu na maioria dos exemplares das classes 1, 2 e 3, consequentemente, os valores médios dos custos totais ($C_1 \times \{\text{nº de Obj. Processados}\} + C_2 \times \{setup\} + \{\text{Desperdício total}\}$) calculados mostram nas três primeiras classes uma diferença acentuada em relação às soluções dadas pelo CPLEX. Por outro lado, nas demais classes o *SA* demonstra competitividade uma vez que a diferença é muita baixa, tanto com

$C_2 = 1$ ou $C_2 = 10$, e superior nas Classes 13, 14 e 16, quando $C_2 = 10$.

Devido ao fato de um alto custo financeiro para adquirir um *solver* comercial, a heurística apresentada torna-se uma ferramenta mais acessível e portanto mais atrativa, pois em grande parte dos experimentos conseguiu obter soluções suficientemente boas. Uma opção para encontrar soluções melhores nas classes com maior diferença seria aumentar número de iterações internas, entretanto, aumentaria o tempo de execução.

Visivelmente, as classes 3, 4, 5, 6, 11 e 12 apresentam um maior tempo de execução, isso deve-se ao fato de que são as classes com maior número de padrões de corte disponíveis. Apesar de não poder haver uma comparação entre os tempos da heurística e o *solver*, seria conveniente implementar o código na linguagem *C++* para que seja possível melhorar o tempo de máquina.

Outra tentativa de melhorar, seria implementar o método de geração de colunas proposto por Gilmore e Gomory [7], assim, gera-se um número menor de padrões de corte melhores e diminui-se consideravelmente o tempo e provavelmente ocasionaria um aumento da qualidade das soluções.

É importante frisar que o *solver CPLEX*, além do alto custo financeiro, possui uma sofisticada linguagem de programação e que vem sendo melhorada desde o seu surgimento, portanto, os algoritmos aqui implementados surgem como uma alternativa mais simplista, mas que ainda podem ser melhorados e mais sofisticados.

Referências Bibliográficas

[1] Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., and STEWART, W. R. Guidelines for designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics 1*, 1 (1995), 293 – 301.

[2] Chen, C., M., H. S., and M., T. W. A simulated annealing heuristic for the one-dimensional cutting stock problem. *European Journal of Operational Research 93*, 3 (1996), 522–535.

[3] Dyckhoff, H. A typology of cutting and packing problems. *European Journal of Operational Research 44*, 2 (1990), 145–159.

[4] Eglese, R. Simulated annealing: A tool for operational research. *European Journal of Operational Research 46*, 3 (1990), 271 – 281.

[5] Eilon, S. Optimizing the shearing of steel bars. *Journal of Mechanical Engineering Science 2*, 2 (1960), pp. 129 – 142.

[6] Gau, T., and Wäscher, G. Cutgen1: A problem generator for the standard one-dimensional cutting stock problem. *European Journal of Operational Research 84*, 3 (1995), 572 – 579.

[7] Gilmore, P. C., and Gomory, R. E. A linear programming approach to the cutting-stock problem. *Operations Research 9*, 6 (1961), pp. 849–859.

[8] Gilmore, P. C., and Gomory, R. E. A linear programming approach to the cutting stock problem-part ii. *Operations Research 11*, 6 (1963), pp. 863–888.

[9] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research 39*, 3 (1991), 378 – 406.

[10] Kantorovich, L. V. Mathematical Methods of Organizing and Planning Production. *Management Science 6*, 4 (1960), 366 – 422.

[11] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), pp. 671 – 680.

[12] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics 21*, 6 (1953), 1087 – 1092.

[13] Metzger, R. W. *Stock Slitting*. Elementary Mathematical Programming, Wiley, 1958.

[14] 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* (1954), pp. 10 – 13.

[15] Salles Neto, L. L. d. *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.

[16] Suliman, S. M. Pattern generating procedure for the cutting stock problem. *International Journal of Production Economics 74*, 1-3 (2001), 293 – 301.

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

Anexos

Algoritmo da Função Simulated Annealing

```
% Simulated Annealing para problema de corte de estoque.
% Juliano da Silva de Souza - ra 079983
 function [TesteNovo xOpt] = Nv_SA(A,dem,desp,x_inicial)

[m,n] = size(A);
x=x_inicial;
% Limite superior para cada variável
for j = 1 : n
    ind = find(A(:,j));
    limx = dem(ind)./A(ind,j);
    UBx(j)=ceil(max(limx));
end
%
% Dados iniciais

M = 1000;
% setup = 100;
%
t0 = 3000;
tf = 0.001;
alpha = 0.95;
%
y=zeros(n,1);
Iw=find(x);
y(Iw)=1;
```

```
%
t = t0;
%
%Algoritmo
%
sobra = A*x - dem;
xtest = x;
xOpt = x;
yOpt = y;
f = sum(x) + sum(y) + M*sum(sobra)
fOpt = f;
ftest = f;
%
L=4*n; (L = 16*n % para o artigo)
%
nmax=-L;
nmaxint=0;
%
while (t > tf)
    naceita=0;
    while ((naceita < L) && (nmax < nmaxint))
            nmax = nmax + 1;
            [x_vz,y_vz,sobra_vz] =tent_nsol_vz(A,x,dem,UBx);
            fviz = sum(x_vz) + sum(y_vz) + M*sum(sobra_vz);
            delta = fviz - f;
            if (delta < 0)
                x = x_vz;
                f = fviz;
                naceita=naceita + 1;
                if (f < fOpt)
                    xOpt=x_vz;
                    fOpt=f;
                    naceita=L;
                end
            else
```

```
                k = rand(1);
                P = exp(-delta/(t));
                if (k <= P)
                    x = x_vz;
                    f = fviz;
                    naceita = naceita + 1;
                end
            end
    end
    t = alpha*t;
end

x1=xOpt;
TesteNovo=fOpt
end
```

Algoritmo para Gerar Soluções Vizinhas

```
% [Nova]Função que gera novas soluções em uma vizinhança da solução atual;
% Juliano da Silva de Souza - ra 079983

function [x_vz, y_vz, sobra_vz]=tent_nsol_vz(A,x,dem,UBx)

[m,n]=size(A);

x_vz=x;

%### Perturbar, aleatoriamente, os valores das variáveis de decisão;
%
while(x_vz == x)
    x_aux = x;
% Determinando a quantidade de índices das variáveis que irão ser perturbadas
    n_base=length(find(x_aux));
```

```
    Ia=fix(0.7*n_base);   % 70% dos índices diferentes de zero.
    Ib=fix(0.9*n_base);   % 90% dos índices diferentes de zero.
    pb=ceil((Ib - Ia)*rand + Ia);

    bx_i=find(x_aux);
    b_num=length(bx_i);

    for k=1:pb
    aleo=round((b_num - 1)*rand + 1 );
    bx_i(aleo)=[];
    b_num=length(bx_i);
    end
    b_num=length(bx_i);
    bx_i=bx_i';
%
%
    nb_num=round(2*rand + 1);    %N^o de variáveis NB que serão perturbadas;
%
%
% ######################
%     % Gerando, aleatoriamente, vetor de índices de variáveis NÃO BÁSICAS;
%
    nb_x_i = zeros(1,nb_num);
    p = zeros(1,n);
    p(1:n) = 1:n;
    for k = n:-1:n-nb_num+1
        q = ceil(k*rand);
        nb_x_i(n-k+1) = p(q);
        p(q) = p(k);
    end
    nb_x_i = sort(nb_x_i);


%     Determinando os índices das variáveis que serão perturbadas
    z_i=zeros(1,b_num);
```

```
    for j=1:b_num
        for k=1:nb_num
            if(nb_x_i(k)==bx_i(j))
            z_i(j)=bx_i(j);
            end
        end
    end
    id=find(z_i);
    bx_i(id)=[];
    x_i=sort([bx_i nb_x_i]);
    num=length(x_i);

    x_aux(x_i(1,:))=round(rand);%round( UBx(x_i(1,:))*rand );
%
%   Testando a factibilidade;
%
    sobra_aux=A*x_aux - dem;
    if(min(sobra_aux) >= 0)
        x_vz = x_aux;
        sobra_vz = sobra_aux;
    else
        while(min(sobra_aux) < 0)
            % Escolher o índice da variável de menor valor;
            xpert=zeros(2,num);
            for col=1:num
                xpert(1,col)=x_aux(x_i(col));
                xpert(2,col)=x_i(col);
            end
            [Y,V]=min(xpert(1,:));
            xk=xpert(2,V);
            %%%%%
            for i=1:m
                if(A(i,xk)== 0)
                    var_x(i)=0;
                else
```

```
                    var_x(i)=floor(sobra_aux(i)/A(i,xk));
                end
            end
            var_xk=-min(var_x);
            LBxk = var_xk + x_aux(xk);
            if(LBxk <= UBx(xk))
                x_aux(xk) = floor(LBxk);
                x_vz = x_aux;
            else
                x_aux(xk)=x(xk);
            end
            sobra_aux = (A*x_aux) - dem;
            x_i(V)=[];
            num=num-1;
        end
            sobra_vz = sobra_aux;
    end
end

% Reorganiza o vetor binário "y"
y_vz=zeros(n,1);
Iw_vz=find(x_vz);
y_vz(Iw_vz)=1;
```

Geração Proposta no Artigo

```
% Função que busca pontos vizinhos em uma vizinhança da solução atual - Artigo;
% Juliano da Silva de Souza - ra 079983

function [x_vz, y_vz, sobra_vz]=sol_vz(A,x,dem,UBx)

[m,n]=size(A);
```

```
x_vz=x;

% Perturbar, aleatoriamente, os valores das variáveis de decisão;
while(x_vz == x)
    x_aux = x;

    x_i=find(x_aux);%
    num=length(x_i);%
    xj=round((num - 1)*rand + 1 );%Determinar, aleatoriamente, os índices
    xk=round((num - 1)*rand + 1 );%das variáveis que serão perturbadas.
 %
    while(xj == xk)
        xk=round((num - 1)*rand + 1 );%%Garantindo distinção.
    end
 %
    x_aux(xk)=round( UBx(x_i(xk))*rand );
    x_aux(xj)=round( UBx(x_i(xj))*rand );
 %
    sobra_aux=A*x_aux - dem;
    if(min(sobra_aux) >= 0)
        x_vz = x_aux;
        sobra_vz = sobra_aux;
    else
        x_aux(xk) = x(xk);
        sobra_aux = A*x_aux - dem;
        for i=1:m
            if(A(i,xj)== 0)
                var_x(i)=0;
            else
                var_x(i)=floor(sobra_aux(i)/A(i,xj));
            end
        end
        var_xj=max(abs(var_x));
        LBxj = var_xj + x_aux(xj);
        x_aux(xj) = floor(rand(1)*(UBx(xj) - LBxj) + LBxj);
```

```
        x_vz = x_aux;
        sobra_vz = (A*x_aux) - dem;
    end
end

% Reorganiza o vetor binário "y"
y_vz=zeros(n,1);
Iw_vz=find(x_vz);
y_vz(Iw_vz)=1;
```

Gerador de Padrões de Corte

```
% Função geradora de colunas para problema de corte de estoque.
% Juliano da Silva de Souza - ra 079983
for cl=1:20
load classe
lp=LLn(cl);
W = classe(lp,1)
m = classe(lp,2)

for i=(lp+1):(lp+m)
    wj(i-lp)=classe(i,1);
    dem(i-lp)=classe(i,2);
end
tic
n=length(wj);
m=length(W);

% Gerando padrões de corte, isto é, colunas da matriz restrição do problema
% original.
for k=1:m;
j=1;

% Encontrar a 1a coluna
```

```
s=0;
A = zeros(n,j,k);
for i=1:n
  s = A(1:(i-1),j,k)'*wi(1:(i-1));
  A(i,j,k)=fix((W(k)- s)/wi(i));
end
%
flag = 1;
t1=5;
while (flag && t1<1800)
    i = find(A(1:n-1,j,k) ~= 0, 1, 'last' );
    j= j + 1;
    A(1:i-1,j,k) = A(1:i-1,j-1,k);
         A(i,j,k)=A(i,j-1,k)-1;
        for z= i+1:n
            s = A(1:z-1,j,k)'*wi(1:z-1);
            A(z,j,k)=fix((W(k)- s)/wi(z));
        end
    clear s
    flag = ~isempty(i);
    t1=toc;
end
% Calcular o vetor c de entradas c_jk correspondente as perdas do
% padrão de corte j na bobina k
for j=1:length(A(1,:,k))
    c(j,k)=W(k)- A(:,j,k)'*wi;
end
end
desp=c;
save(['A',num2str(cl),'.mat'])
clear all
end
```