Proceeding Series of the Brazilian Society of Applied and Computational Mathematics, Vol. 3, N. 1, 2015.
Trabalho apresentado no XXXV CNMAC, Natal-RN, 2014.

Um Simulated Annealing *no Problema do Corte Unidimensional Inteiro*

Angelo Aliano Filho **Antônio Carlos Moretti**
Instituto de Matemática, Estatística e Computação Científica - IMECC
Universidade Estadual de Campinas - UNICAMP
CEP 13083-859, Campinas, SP

Resumo: *O presente trabalho trata o Problema do Corte Unidimensional Inteiro (PCUI). Devido a sua vasta aplicabilidade nas metalúrgicas, indústrias textil, móveis, papel, alumínio, etc, este problema é um dos mais estudados em Otimização Combinatória. Embora seja facilmente entendido, o PCUI é altamente complexo de ser resolvido, devido a natureza e o número de variáveis envolvidas mesmo para pequenas instâncias, tornando inviável a aplicação direta dos algoritmos exatos do tipo* Branch*. Por este motivo, aplicou-se a metaheurística Simulated Annealing (SA), com alguma modificações em relação ao proposto em [10], para resolvê-lo. Para tanto, duas heurísticas para (i) construir padrões de corte e (ii) soluções admissíveis para o problema, foram desenvolvidas. Para validá-las, gerou-se 1800 problemas testes e comparou-se a qualidade das soluções do método proposto com a solução do Problema Linear Inteiro (PLI) obtido com os padrões gerados pelo consagrado método de* Geração de Colunas *(GC) desenvolvido por [5]. Os resultados mostraram uma boa performance e robustez do método proposto, obtendo soluções sub-ótimas num razoável tempo computacional.*

Palavras-chave: *Otimização Combinatória, Problema de Corte,* Simulated Annealing

1 Introdução

O PCUI é um dos problemas combinatórios mais estudados, devido, principalmente, a sua aplicabilidade no mundo da engenharia de produção, fazendo parte do planejamento de uma diversidade de indústrias que cortam papel, móveis, vidro, plásticos, tecido, entre outras. Apesar de ser muito simples de ser entendido, este problema tem um elevado nível de complexidade, sendo classificado na literatura como $\mathcal{NP}$-difícil [6].

Basicamente, o problema é entendido da seguinte forma: as matérias primas utilizadas são inicialmente produzidas em tamanhos grandes já padronizados, possivelmente estocadas e, somente mais tarde, reduzidas a tamanhos menores para então serem usadas pela indústria de acordo com as demandas externas, muitas vezes não padronizadas. No entanto, esta necessidade causa o inconveniente de preparar as máquinas para a operação de corte destes materiais, ocasionando, inevitavelmente, o aumento do tempo de produção. Nesse sentido, há necessidade de se planejar estes cortes, a fim de minimizar o desperdício e efeitos negativos ocasionados com esta operação.

A tarefa de se obter uma solução ótima exata para o PCUI utilizando métodos clássicos de otimização é desafiante. Isso se deve, principalmente, à integrabilidade e o elevado número de variáveis decisórias envolvidas neste problema. Alguns poucos algoritmos exatos para encontrar a solução ótima inteira do PCUI são conhecidos na literatura, como [9], [3], [12] e [11]. No entanto, estes métodos podem ser aplicados apenas a problemas de pequeno porte.

Sendo assim, metodologias aproximativas têm sido desenvolvidas nas últimas cinco décadas para este problema. O primeiro trabalho a tratar o PCUI de maneira não exata foi [5], que propuseram uma técnica de GC para obtenção de uma solução ótima contínua. Outros métodos

DOI: 10.5540/03.2015.003.01.0406

não-exatos têm sido desenvolvidos e uma breve revisão é dada em [4]. Ressalta-se também o uso de Heurísticas de Arredondamento, proposto por [1], e algumas metaheurísticas específicas, desenvolvidas por [2], [7], [8], entre tantos.

Mesmo que a otimalidade não seja garantida, o objetivo deste trabalho é apresentar um método heurístico de solução alternativo que forneça soluções "boas" e rápidas para este problema.

2 Modelagem Matemática

Para modelar o PCUI, considere uma objeto em estoque, de tamanho L (comprimento padrão), e m o número de itens demandados. Cada item i tem comprimento $l_i < L$ e ao menos d_i unidades precisam ser produzidas a fim de atender a demanda necessária, $i = 1, ..., m$. O objetivo consiste em minimizar o número de objetos a serem cortados a fim de atender à esta demanda mínima exigida. Do ponto de vista operacional, aqui apenas faz sentido cortar um número inteiro de vezes desta bobina mestre, o que torna este problema difícil de ser resolvido.

Uma instrução que lista os itens demandados a serem cortados do objeto em estoque é chamada de *padrão de corte*, que pode ser associado ao vetor m-dimensional

$$\mathbf{a}_j = (a_{1j}, a_{2j}, ..., a_{mj})^T,$$

onde cada entrada $a_{ij} \in \mathbb{N}$, denota a quantidade do item i presente no padrão j, $i = 1, ..., m$.

Definição 2.1. *Um padrão de corte $\mathbf{a}_j \in \mathbb{N}^m$ é admissível se satisfaz as seguintes condições:*

$$\sum_{i=1}^{m} l_i \cdot a_{ij} \leq L, \tag{1}$$

$$\sum_{i=1}^{m} l_i \cdot a_{ij} \geq L - \Delta, \tag{2}$$

$$\sum_{i=1}^{m} a_{ij} \leq F, \tag{3}$$

onde $\Delta = \min_{1 \leq i \leq m} \{l_i\}$ e F é o número de facas máximo permitido para produzí-lo.

Suponha que existam n maneiras distintas de se formar todas as combinações de corte possíveis atendendo as restrições (1)-(3). Como visto, mesmo para m da ordem de dezenas, o valor de n pode facilmente atingir a casa dos milhões ou bilhões, desde que a razão L/l_i fique suficientemente pequena.

O modelo para o PCUI é apresentado a seguir:

$$\text{Minimize} \quad z = \sum_{i=1}^{n} x_j \tag{4}$$

$$\text{sujeito a} \quad \sum_{i=1}^{m} a_{ij} \cdot x_j \geq d_i,\ i = 1, ...m, \tag{5}$$

$$x_j \in \mathbb{N},\ j = 1, ..., n. \tag{6}$$

A variável decisória deste problema é x_j, e denota a frequência do padrão j a ser utilizado, $j = 1, ..., n$. O conjunto de restrições (5) diz respeito ao atendimento mínimo da demanda exigida para cada item i.

Como salientado na seção anterior, este modelo tem dois agravantes, e que dificultam a obtenção da solução ótima exata $\mathbf{x}^* = (x_1^*, x_2^*, ..., x_n^*)^T$: (i) as condições de integrabilidade (6) e

(ii) o elevado valor de n, o que torna inviável resolver o PLI (4)-(6) usando os métodos exatos clássicos como *Branch-and-{Bound,Cut,Price}*.

A fim de encontrar um limitante inferior $\underline{z}$ para o valor da função objetivo e avaliar a qualidade das soluções heurísticas, aplicou-se o algoritmo clássico GC e em seguida resolveu o PLI com as colunas de Gilmory-Gomory encontradas. Por simplicidade, chama-se este procedimento de obter $\underline{z}$ de "Quase-Exato"(QE), em virtude da boa qualidade deste limitante em relação ao original.

3 O método SA

Em virtude da elevada complexidade do PCUI e na impossibilidade de solucioná-lo via métodos exatos, apresenta-se uma metaheurística alternativa, a fim de obter soluções admissíveis que aproximam $\mathbf{x}^*$ num tempo computacional aceitável. Para isto, foram desenvolvidas duas heurísticas: uma para construir padrões admissíveis, respeitando as restrições (1)-(3) e a outra para construir soluções factíveis para o problema atendendo (5).

A codificação de uma solução $\mathbf{S}$ para o PCUI é composta por uma matriz $\mathbf{A} \in \mathbb{N}^{m \times m}$, onde a coluna $\mathbf{a}_j$ é o j-ésimo padrão de corte, acoplada um vetor linha $\mathbf{x}^T \in \mathbb{N}^{1 \times m}$, cuja componente j indica a frequência do padrão $\mathbf{a}_j$ nesta solução, ou seja, $\mathbf{S} = \left[\frac{\mathbf{A}}{\mathbf{x}^T}\right]$.

Sendo assim, esses algoritmos fazem com que o método desenvolvido busque uma solução para o problema somente no espaço admissível. Os passos destes dois algoritmos são apresentados na próxima subseção.

3.1 Heurísticas construtivas

O primeiro procedimento constroi padrões admissíveis e tem como dados de entrada m, F, l_i, L. Para evitar um laço infinito, um vetor $\mathbf{v}$ é inicializado, e indica a ordem dos itens que têm preferência de escolha para entrar no padrão a ser formado. Um pseudocódigo deste algoritmo é apresentado na Figura (1).

Procedimento 1: construção de padrões

Input: $\{m, F, l_i, L\}$
$Sobra = L$
Enquanto $Sobra > \Delta = \min_{1 \le i \le m} \{l_i\}$
  Inicialize $\mathbf{v}$
  Para $i = 1$ até m
    Tome o i-ésimo item de $\mathbf{v}$
    Calcule a_{ij}: quantas vezes este item pode ser adicionado em uma fração da sobra deste padrão
    Se o número de facas não foi excedido, aloque a_{ij} na posição correspondente **Fim-Se**
    Atualize $Sobra = L - \sum_{k=1}^{i} l_k \cdot a_{kj}$
  Fim-Para
Fim-Enquanto
Output: $\mathbf{a}_j$

Figura 1: Algoritmo para construir padrões admissíveis

Apresenta-se na Figura (2) o procedimento que constroi soluções admissíveis para o problema.

DOI: 10.5540/03.2015.003.01.0406

Procedimento 2: construção de soluções

Input: $\{m,\ F,\ l_i,\ L,\ \mathbf{d}\}$
$\mathbf{S} = \varnothing$
producao $= 0$
Falta $=$ **producao** $-\ \mathbf{d}$
Enquanto Falta < 0
Certifique os itens que ainda faltam para completar a demanda
Aloque, nas primeiras posições do vetor $\mathbf{v}$, os itens cujas demandas não foram atendidas
Construa um padrão $\mathbf{a}_j$ utilizando o **Procedimento 1**
Calcule

$$x_j = \min_i \left\{ \left\lceil \frac{-Falta_i}{a_{ij}} \right\rceil : a_{ij} \neq 0 \right\}$$

producao $=$ **producao** $+\ x_j \cdot \mathbf{a}_j$
Falta $=$ **producao** $-\ \mathbf{d}$
$\mathbf{S} = \mathbf{S} \cup \left\lceil \dfrac{\mathbf{a}_j}{x_j} \right\rceil$
Fim-Enquanto
Output: $\mathbf{S}$

Figura 2: Algoritmo para construir soluções admissíveis

Este algoritmo constroi soluções admissíveis ao PCUI quanto às restrições de demanda e que contenham, no máximo, m padrões. Caso a demanda tenha sido atendida mesmo antes de terem usado as m colunas de $\mathbf{S}$, completa-se a dimensão desta matriz com colunas nulas. Isto pode ser útil quando se interessa em soluções que utilizem um menor *setup*.

O método SA desenvolvido difere do original, proposto por [13], no tocante ao número de soluções consideradas no processo de pesquisa. Na abordagem original, o SA inicia a sua busca com uma única solução inicial e temperatura T_0 e a partir dela, um vizinho é construído. Então um movimento é realizado se o vizinho é promissor ou com uma probabilidade, tanto menor quanto maior for a diferença entre os valores objetivos e menor a temperatura atual. Depois de transcorridas algumas iterações em uma isoterma, a temperatura atual T é multiplicada por uma constante α (taxa de arrefecimento) e o algoritmo prossegue novamente, até que a temperatura atinja o valor $T_f \approx 0$.

O que difere a versão original da proposta neste trabalho, é o número de soluções consideradas na pesquisa. A ideia consiste em combinar estas soluções, fazendo-as iteragir entre si, garantindo uma pesquisa sobre o espaço de busca que seja independente da solução inicial, mais global e eficaz. Então, inicia-se o SA com β soluções usando-se os **Procedimentos 1** e **2** e uma dada temperatura inicial T_0. Em seguida, a solução atual $\mathbf{S}^k$, $k = 1, ..., \beta$ é inicializada e, avaliada segundo a função f

$$f(\mathbf{S}^k) = \sum_{i=1}^{m} x_j + 10 \cdot \left(\sum_{i=1}^{m} \sum_{j=1}^{m} a_{ij} \cdot x_j - d_i \right), \quad k = 1, ..., \beta, \tag{7}$$

que prioriza as soluções que têm a menor super-produção possível.

Um vizinho é construído da seguinte forma: dada a solução atual $\mathbf{S}^k$, um vizinho é uma outra solução $\mathbf{V}^k$ que possui os $m-1$ padrões $\mathbf{a}_j$ da original, e o remanescente é gerado pelo **Procedimento 1**. Nesse caso, o vetor $(\mathbf{x}^k)^T$ da solução vizinha é recalculado.

Se $f(\mathbf{V}^k) < f(\mathbf{S}^k)$ então a solução atual é atualizada, ou seja, realiza-se um movimento. Caso contrário, o movimento é realizado com probabilidade

$$\varphi = e^{-\Delta/T} \tag{8}$$

onde T é a temperatura do sistema e

$$\Lambda = 0,75 \cdot \frac{f(\mathbf{V}^k) - f(\mathbf{S}^k)}{f(\mathbf{S}^k)} + 0,25 \cdot \frac{f(\mathbf{V}^k) - \min\limits_{1 \leq k \leq \beta} f(\mathbf{S}^k)}{f(\mathbf{S}^k)}, \quad k = 1, ..., \beta. \tag{9}$$

O cálculo de Λ, é uma combinação convexa entre as diferenças relativas da solução atual com a sua vizinha e a melhor até então encontrada. A ideia é levar em conta informações entre as β soluções que realizam a pesquisa e fazê-las redescobrir outros espaços mais promissores para o problema, de maneira mais rápida e inteligente.

Em uma isoterma, foi permitido, no máximo, θm movimentos. Após isto, diminui-se a temperatura, isto é, $T \leftarrow \alpha \cdot T$. O algoritmo se encerra quando a temperatura atinge o nível $T_f \approx 0$. Foi implementado também o processo de *re-annealing*, que nada mais é do que reaquecer o sistema após ter sido congelado. Foi permitido σ *re-annelings* no SA proposto.

A seção dos testes computacionais demonstra a eficiência computacional do SA, e como foi capaz de obter soluções factíveis de ótima qualidade.

4 Testes Computacionais

Os testes computacionais e os algoritmos desenvolvidos para este problema foram codificados usando o software Matlab, versão 7.10.0 R2010a, em um computador Core 2 Quad com 3GB de memória, do Instituto de Matemática, Estatística e Computação Científica - IMECC/UNICAMP, Campinas, SP, Brasil. Nestes experimentos, tem-se 18 classes para o PCUI. Para cada classe foram gerados 100 exemplares usando o gerador [14]. A fim de avaliar o método proposto, cada um dos 1800 exemplares foram resolvidos uma vez pelo procedimento QE e 20 vezes pelo SA. Assim, comparou-se a solução QE com a média dos resultados obtidos com o SA.

As 18 classes contemplam todas as combinações possíveis entre itens pequenos, médios e grandes, demandas baixas e altas e quantidade de itens. Assim, os valores para l_i foram gerados entre $[v_1 L, v_2 L]$, onde $L = 10.000$ foi fixo. O número de facas foi definido como $F = \left\lceil \sum_{i=1}^{m} \frac{L}{m \cdot l_i} \right\rceil$. Além disto, a demanda d_i para cada item foi obtida gerando-se valores com distribuição normal com média $\bar{d}m$. As características de cada classe são mostradas na Tabela (1).

Tabela 1: Classes para o PCUI

Classe	m	v_1	v_2	$\bar{d}$	Classe	m	v_1	v_2	$\bar{d}$
1	10	0,01	0,2	10	10	20	0,01	0,8	50
2	10	0,01	0,2	50	11	20	0,2	0,8	10
3	10	0,01	0,8	10	12	20	0,2	0,8	50
4	10	0,01	0,8	50	13	40	0,01	0,2	10
5	10	0,2	0,8	10	14	40	0,01	0,2	50
6	10	0,2	0,8	50	15	40	0,01	0,8	10
7	20	0,01	0,2	10	16	40	0,01	0,8	50
8	20	0,01	0,2	50	17	40	0,2	0,8	10
9	20	0,01	0,8	10	18	40	0,2	0,8	50

A Tabela (2) ilustra os parâmetros empregados para o SA para as classes 1 a 6, 7 a 12 e 13 a 18.

A seguir, apresenta-se os valores da função objetivo z_{SA} e $\underline{z}$, encontrados pelo SA e pelo QE, respectivamente. O PLI com as colunas de Gilmore-Gomory foi resolvido pelo mesmo software empregado no desenvolvimento dos algoritmos heurísticos, limitando-se o número máximo de nós pesquisados no algoritmo *Branch-and-Bound* em $1.000 \times m$.

Computou-se os valores para $\bar{t}$, $\bar{y}$ e $\bar{\gamma}$ e $\bar{w}$ onde representam os valores médios, respectivamente, do tempo médio de processsamento (em seguntos), do *setup* da máquina, do *gap* relativo

Tabela 2: Paramêtros para o SA

Classe	α	β	θ	σ	T_0	T_f
1 a 6	0,90	2	1	2	10	10^{-4}
7 a 12	0,92	4	1,5	2	100	10^{-6}
13 a 18	0,94	6	2	1	1.000	10^{-8}

(em %) dos valores objetivos entre o SA e do QE e do número de iterações necessários[1]. Os números $\bar{r}$ e $\bar{s}$ correspondem à super-produção em relação à quantidade demandada e a perda relativa associada (em %).

Os indices "h" e "q" que aparecem em t, y, r, s e w fazem menção ao tipo de método empregado: "h" está associado ao SA e "q" com o QE.

Tabela 3: Resultados numéricos médios para as 18 classses do PCUI

Classe	$\bar{t}_h$	$\bar{t}_q$	$\bar{z}_{SA}$	$\underline{z}$	$\bar{y}_h$	$\bar{y}_q$	$\bar{r}_h$	$\bar{r}_q$	$\bar{s}_h$	$\bar{s}_q$	$\bar{w}_h$	$\bar{w}_q$	$\bar{\gamma}$
1	29,8	**5,3**	144,2	**143,5**	**7,6**	9,3	**0,15**	0,19	**7,4**	14,1	167	20,1	0,48
2	28,7	**6,7**	502,3	**498,7**	**8,3**	9,6	5,2	**3,2**	15,5	**10,3**	167	22,4	0,72
3	25,4	**4,8**	538,4	**534,5**	**7,3**	9,5	**10,9**	11,8	**10,8**	9,7	167	16,3	0,94
4	29,3	**5,3**	2670,4	**2658,7**	**7,7**	9,6	17,9	**15,7**	**7,8**	17,5	167	13,5	0,45
5	27,4	**4,1**	613,1	**610,5**	**8,4**	9,6	**8,4**	9,7	**5,1**	12,6	167	11,2	0,49
6	28,8	**3,6**	3521,5	**3520,1**	**8,6**	9,8	9,3	**8,7**	17,9	**17,8**	167	9,8	0,00
7	**122,3**	415,6	615,4	**610,8**	**18,6**	19,7	**2,1**	3,2	**0,1**	0,8	300	34,3	0,86
8	**125,2**	351,3	2176,4	**2155,7**	**18,7**	19,1	**0,4**	2,4	**0,1**	1,1	300	32,5	0,97
9	**121,4**	318,9	1509,2	**1489,3**	**17,3**	19,6	7,8	**6,9**	6,8	**0,8**	300	31,3	1,30
10	**127,8**	326,3	8877,4	**8840,2**	**17,9**	18,9	13,3	**10,9**	10,5	**0,6**	300	29,4	0,42
11	**131,5**	289,5	1981,3	**1962,4**	**16,2**	19,4	**0,3**	1,3	**1,8**	3,6	300	27,9	0,97
12	**126,6**	258,4	11548,2	**11395,4**	**17,9**	19,4	**8,8**	9,3	**7,1**	9,2	300	27,4	1,34
13	**552,6**	835,2	1764,2	**1759,4**	**38,8**	39,4	**1,1**	1,2	**0,1**	0,5	672	86,3	0,28
14	**535,2**	979,3	9278,9	**9227,8**	**38,4**	39,7	1,4	**0,2**	**0,4**	0,6	672	84,7	0,55
15	**518,3**	762,1	8189,4	**8069,3**	**34,7**	39,3	13,7	**8,6**	12,0	**8,8**	672	71,5	1,49
16	**545,4**	793,1	36709,2	**36055,4**	**35,5**	39,6	**4,6**	6,6	**1,4**	4,8	672	76,1	1,81
17	**523,2**	767,3	10153,4	**10050,3**	**37,1**	38,8	12,3	**6,8**	13,5	**5,5**	672	74,6	1,03
18	**539,6**	883,9	53056,7	**53047,2**	**36,2**	38,4	10,4	**7,1**	**5,1**	15,2	672	78,8	0,00

Pelos resultados apresentados na Tabela (3), pode-se concluir que a metaheurística proposta foi eficaz. Isto pode ser afirmado comparando-se os valores de $\bar{\gamma}$. Para todas as classes geradas, obteve-se soluções sub-ótimas de excelentes qualidades num tempo computacional razoável, com $\bar{\gamma}$ inferior a $2,0\%$.

Nitidamente houve uma melhor performance do SA naquelas classes onde os itens são pequenos e grandes. Nas classes onde os itens tem tamanhos variados, houve uma queda de eficiência no SA, ou seja, estas classe são mais difíceis de serem resolvidas heuristicamente.

O método desenvolvido foi robusto. O tempo transcorrido para resolver o PCUI utilizando-se este SA quadruplicou-se com m, aproximadamente. No QE, em virtude do PLI obtido, os exemplares com $m = 20$ e $m = 40$ foram resolvidos mais lentamente que o SA.

Outras características importantes dos resultados obtidos foram o *setup*, a super-produção e o desperdício gerado pelas soluções do SA, que levam vantagem sobre a solução da técnica QE na maioria dos testes. Como se pode ver, em todos os 18 tipos, o número médio de padrões utilizados do SA foi menor do que a do método concorrente. Em relação à super-produção, nove classes resolvidas pelo SA superam o QE. Quanto ao desperdício, o SA foi melhor em 12 classes. Isso pode ser explicado pelas heurísticas construtivas e pela função de avaliação (7), permitindo-se construir soluções que atendem à demanda, com a menhor super-produção possível, utilizando um número mínimo e "bons" de padrões de corte. Do ponto de vista operacional e prático, soluções com esta características são vantajosas em relação às outras, pois o excesso de produção é vendido à preços inferiores e a troca de máquinas consome tempo e ajustes.

[1]No caso do QE, w_q é o número de iterações necessários para o GC.

DOI: 10.5540/03.2015.003.01.0406

5 Conclusões

Este trabalho visou solucionar o PCUI utilizando a metaheurística *Simulated Annealing* com duas herísticas construtivas. A fim de comparar e avaliar a solução da metaheurística, implementou-se o clássico algoritmo de Geração de Colunas, e em seguida, resolveu-se o Problema Linear Inteiro com as colunas obtidas, usando-se o *Branch-and-Bound*. Os resultados demostraram eficiência e robustez do método aqui desenvolvido, tendo em vista a qualidade das soluções encontradas e o tempo de processamento. Além disso, o método implementado obteve soluções mais interessantes que as fornecidas pelo concorrente, no tocante à superprodução, ao número de padrões utilizados e o desperdício. Isto pode ser entendido como um sub-produto do método apresentado, sendo uma ferramenta simples, fácil de ser implementada e do ponto de vista computacional, viável.

Como perspectiva futura de novos trabalhos, propõe-se utilizar estes algoritmos heurísticos para tratar o PCUI com múltiplos objetivos.

Agradecimentos

Este trabalho tem apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo, FAPESP, processo 2013/06035-0.

Referências

[1] K. C. Poldi & M. N. Arenales. Heurísticas para o Problema do Corte Unidimensional Inteiro. *Pesquisa Operacional*, 26:473–492, 2006.

[2] H. Gehring & A. Bortfeldt. A Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operations Research*, 4:401–418, 1998.

[3] J. M. V. Carvalho. Exact Solution of Bin-packing Problems Using Column Generation and Branch-&-bound. *Annals of Operations Research*, 86:629–659, 1999.

[4] G. Wascher & T. Gau. Heuristic for the Integer One-dimensional Cutting Stock Problem: a Computational Study. *OR Spektrum*, 18:131–144, 1996.

[5] P. C. Gilmore & R. E. Gomory. A Linear Programming Approach to the Cutting-Stock Problem. *Oper Res*, 9:848–859, 1961.

[6] E. Coffman M. Garey & D. Johnson. *Approximation Algorithms for Bin Packing: A Survey.* Boston. PWS, 1996. Approximation algorithms for NP-hard problems.

[7] R. R. Golfeto & A. C. Moretti & L. L. N. Sales. A Grasp Metaheuristic for the Ordered Cutting Stock Problem. *Revista Chilena de Ingeniería (En línea)*, 16:421–427, 2008.

[8] R. R. Golfeto & A. C. Moretti & L. L. N. Sales. A Genetic Symbiotic Algorithm Applied to the One-dimensional Cutting Stock Problem. *Pesquisa Operacional (impresso)*, 9:365–382, 2009.

[9] G. Belov & G. Scheithauer. Branch-&-Price Algorithms for the One-Dimensional Cutting Stock Problem. *European Journal of Research Operational*, pages 85–106, 2006.

[10] C-L. S. Chen & S. M. Hart & W. M. Tham. A Simulated Annealing Heuristic for the One-dimensional Cutting Stock Problem. *European Journal of Operational Research*, 93:422–535, 1996.

[11] P. H. Vance. A Branch-&-Cut-&-Price Algorithm for One-Dimensional Stock Cutting and Two-Dimensional Two-Stage Cutting. *Computational Optimization and Applications*, 9:211–228, 1998.

[12] F. Vanderbeck. An Exact Algorithm for ip Column Generation. *Operations Research Letters*, 19:151–159, 1996.

[13] S. Kirkpatrick & D. C. Gellat & M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[14] T. Gau & G. Wascher. CUTGEN: A Problem Generator for the Standard One-dimensional Cutting Stock Problem. *European Journal of Research Operational*, pages 572–579, 1995.

DOI: 10.5540/03.2015.003.01.0406