SPOLM 2007

ISSN 2175-6295 Rio de Janeiro- Brasil, 08 e 09 novembro de 2007.

MÉTODOS HEURÍSTICOS BASEADOS EM GRASP, ILS E VND APLICADOS À RESOLUÇÃO DO PROBLEMA DE SEQÜENCIAMENTO EM UMA MÁQUINA COM PENALIDADES POR ANTECIPAÇÃO E ATRASO DA PRODUÇÃO

Aloísio de Castro Gomes Júnior, Carlos Roberto Venâncio de Carvalho
Universidade Federal de Minas Gerais - UFMG
Av. Presidente Antônio Carlos, 6627, CEP 30161-010, Belo Horizonte - MG
algomesjr2004@yahoo.com.br, carlos@dep.ufmg.br

Pablo Luiz Araújo Munhoz, Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto - UFOP
ICEB, Departamento de Computação, CEP 35400-000, Ouro Preto-MG
pablo.munhoz@gmail.com, marcone@iceb.ufop.br

Resumo

Este trabalho trata do problema de seqüenciamento em uma máquina com penalidades por antecipação e atraso da produção, considerando janelas de entrega e tempo de preparação da máquina dependente da seqüência de produção. Propõe-se um modelo de programação linear inteira mista para representar o problema, bem como dois procedimentos heurísticos de resolução baseados em GRASP, *Iterated Local Search* e *Variable Neighborhood Descent*. Para cada seqüência de *jobs* gerada pelas heurísticas propostas, usa-se um algoritmo de tempo polinomial para determinar a data ótima de início de processamento dos *jobs* na seqüência dada. Experimentos computacionais realizados sobre um conjunto de problemas-teste gerados aleatoriamente indicam que os métodos propostos conseguem alcançar o valor ótimo na maioria dos problemas de pequenas dimensões (8 a 12 *jobs*) e em problemas de dimensões maiores (15 a 75 *jobs*) obtêm desvios baixos em relação à melhor solução obtida.

Palavras-Chaves: Seqüenciamento em uma máquina; Metaheurísticas; Iterated Local Search, GRASP, Descida em Vizinhança Variável.

Abstract

This paper deals with the problem of scheduling single machine with earliness and tardiness penalties, with due windows and sequence dependent setup. A model of mixed integer linear programming is proposed to represent the problem, as well two heuristics based on GRASP, Iterated Local Search and Variable Neighborhood Descent to solve it. For each job sequence generated by the heuristics, an optimal timing algorithm is used to determine the completion time for each job in the job sequence. Computational experiments realized with instances randomly generated show that the methods proposed are able to reach the optimal solution in most of the small instances (8 to 12 jobs) and yield small gaps in instances with 15 to 75 jobs.

Keywords: Single Machine Scheduling; metaheuristics; Iterated Local Search; GRASP, Variable Neighborhood Descent.

1. INTRODUÇÃO

A produção de bens sob encomenda vem sendo adotada por um grande número de empresas, as quais buscam adaptar sua linha de produção às necessidades do cliente. O objetivo é atender ao cliente da melhor forma possível, tanto com relação à quantidade de produtos requeridos quanto em relação à data de entrega. Assim, uma das mais importantes decisões relativas ao chão de fábrica que essas empresas têm que tomar diz respeito à programação da produção.

O problema de programação da produção consiste basicamente em, dado um período do horizonte de planejamento (tipicamente uma semana), fazer a alocação de operações a máquinas e a programação dessas operações em cada máquina, ou seja, decidir a seqüência nas quais as operações devem ser processadas e em que momento elas devem ser realizadas no período de planejamento.

Os elementos comuns desta classe de problema são:

(*i*) Recursos: Um bem ou serviço, cuja disponibilidade é limitada ou não. Alguns exemplos mais comuns de recursos são: máquinas, matérias-primas, mão-de-obra, entre outros.

(*ii*) Tarefas ou Operações: Trabalho elementar cuja realização necessita de certa quantidade de tempo e/ou recursos.

(*iii*) *Job*: seqüência conhecida de uma ou mais operações que representam uma seqüência tecnológica de fabricação de um produto.

Entre os principais tipos de problemas de programação da produção encontra-se o de seqüenciamento em uma máquina com penalidades por antecipação e atraso na produção (PSUMAA – *Single Machine Scheduling for Minimizing Earliness and Tardiness Penalties*). Resolver este tipo de problema pode implicar em uma sensível redução dos custos gerados com a antecipação e com o atraso na produção. Há antecipação quando a produção é finalizada antes da data desejada para a entrega do produto, gerando-se estoques. Por outro lado, há atraso quando a produção é finalizada após a data desejada para a entrega do produto, incorrendo-se em multas. As datas de entrega podem ser comuns ou distintas a todas as operações de produção. Uma formulação mais geral para este tipo de problema considera uma janela de tempo, denominada janela de entrega, onde as operações que forem finalizadas dentro desse intervalo de tempo não implicam em custos adicionais à empresa.

Geralmente outras restrições são impostas a este tipo de problema. Uma dessas restrições adicionais é com relação ao tempo de preparação da máquina. Esse tempo, denominado tempo de *setup*, inclui o tempo de preparar a máquina, o processo ou a oficina para a fabricação de determinados tipos de produtos.

Vários trabalhos encontrados na literatura têm seu foco em técnicas para a resolução do PSUMAA. Este fato se deve à dificuldade de resolução exata do problema, em vista de o PSUMAA pertencer à classe de problemas NP-difíceis [1, 9, 12 e 17] e ao enorme número de aplicações práticas existentes em indústrias metalúrgicas, têxteis, de tintas, entre outras. Uma aplicação em indústrias siderúrgicas, por exemplo, é a fabricação de fios metálicos, onde o conjunto de laminadores é considerado como uma única máquina [2].

Alguns desses trabalhos utilizam métodos de enumeração para resolução do PSUMAA, como, por exemplo, [11 e 12] (PSUMAA sem tempo ocioso de máquina), [3] (PSUMAA com datas distintas de entrega) e [16] (PSUMAA com datas comuns de entrega e tempo de preparação da máquina dependente da seqüência de produção). Dada a dificuldade de encontrar a melhor programação da programação, para problemas de dimensões maiores a abordagem mais comum para resolvê-los é por meio de métodos heurísticos.

[17] apresentam um método baseado na metaheurística Busca Tabu (BT), proposta inicialmente por [6] e [8], para resolver o PSUMAA com janelas de entrega distintas. Os autores dividem o problema em dois subproblemas: determinar a seqüência dos *jobs* e determinar a data de conclusão ótima do processamento de cada *job*. Para resolver o primeiro subproblema, os autores desenvolvem um método baseado na metaheurística BT. Já para

resolver o segundo subproblema os autores propõem um algoritmo de complexidade polinomial para determinar a data de conclusão ótima, explorando as características do problema.

[9] apresentam métodos baseados nas metaheurísticas BT e Algoritmos Genéticos (AG) para resolver o PSUMAA com datas comuns de entrega. Os autores não consideram tempos ociosos de máquina. O mesmo problema foi resolvido por [4], mas aplicando algoritmos baseados nas metaheurísticas BT e *Simulated Annealing* (SA).

[10] aplicam a metaheurística AG ao PSUMAA com datas de entrega distintas e, assim como [17], dividem o problema em dois subproblemas. Para determinar a seqüência de *jobs* utilizam a metaheurística AG e para determinar a data ótima de conclusão do processamento de cada *job* utilizam um algoritmo específico, de complexidade polinomial.

[7] propõem dois métodos para a resolução do problema de seqüenciamento em uma máquina com minimização do tempo total de atraso e tempo de preparação de máquina dependente da seqüência de produção. Um destes métodos baseia-se na metaheurística GRASP, proposta inicialmente por [5], e o outro no método heurístico "*space-based local search*". O método GRASP, proposto por [7], é dividido em três fases: Construção, Melhoramento e Reconexão por Caminhos.

Neste trabalho é apresentado um modelo de programação linear inteira mista (PLIM) para representar o PSUMAA com janelas de entrega e tempo de preparação da máquina dependente da seqüência de produção. Em seguida são propostos dois métodos heurísticos híbridos para resolver o problema proposto. O primeiro método é baseado em GRASP e *Variable Neighborhood Descent* (VND) (proposto por [15]), denominado GRASP-VND-RT. Já o segundo método é baseado em GRASP, *Iterated Local Search* (ILS) e VND, denominado GILS-VND-RT. Para cada seqüência de *jobs* gerada pelos métodos é acionado um procedimento de complexidade polinomial para determinar a data ótima de início de processamento de cada *job* da seqüência dada. Os resultados obtidos pelos métodos propostos são comparados com aqueles obtidos pela resolução do modelo de PLIM usando o otimizador CPLEX, versão 9.1.

Este trabalho está organizado como segue. Na seção 2 são descritas, em detalhes, as características do problema estudado, enquanto na seção 3 é apresentado o modelo de programação linear inteira mista para representar o mesmo. Na seção 4 é apresentado o algoritmo utilizado para determinar a data ótima de início de processamento de cada *job*, enquanto na seção 5 são desenvolvidas as metodologias heurísticas para a resolução do PSUMAA. Na seção 6 são apresentados e discutidos os resultados computacionais e na seção 7 conclui-se o trabalho.

2. DESCRIÇÃO DO PROBLEMA ESTUDADO

O problema estudado neste trabalho é o PSUMAA com tempo de preparação da máquina dependente da seqüência de produção e janelas de entrega e possui as seguintes características:

(*i*) Uma máquina deve processar um conjunto de *n jobs*.

(*ii*) Cada *job* possui um tempo de processamento P_i, uma data inicial E_i e uma data final T_i desejadas para o término do processamento.

(*iii*) A máquina executa no máximo um *job* por vez e uma vez iniciado o processamento de um *job*, o mesmo deve ser finalizado, ou seja, não é permitida a interrupção do processamento.

(*iv*) Todos os *jobs* estão disponíveis para processamento na data 0.

(*v*) Quando um *job j* é seqüenciado imediatamente após um *job i*, sendo estes pertencentes a diferentes famílias de produtos, é necessário um tempo S_{ij} para a preparação da máquina. Tempos de preparação de máquina nulos ($S_{ij} = 0$) implicam em produtos da mesma família. Assume-se, ainda, que a máquina não necessita de tempo de preparação inicial, ou seja, o tempo de preparação da máquina para o processamento do primeiro *job* na seqüência é igual a 0.

(*vi*) É permitido tempo ocioso entre a execução de dois *jobs* consecutivos.

(*vii*) Os *jobs* devem ser finalizados dentro da janela de tempo $[E_i, T_i]$, denominada janela de entrega. Se o *job i* for finalizado antes de E_i então há um custo de manutenção de estoque. Caso o *job* seja finalizado após T_i então há associado um custo por atraso. Os *jobs* que forem finalizados dentro da janela de entrega não proporcionarão nenhum custo para a empresa.

(*viii*) Os custos unitários por antecipação e atraso da produção são dependentes dos *jobs*, ou seja, cada *job i* possui um custo de antecipação α_i e um custo de atraso β_i.

(*ix*) O objetivo a ser alcançado com a resolução deste problema é a minimização do somatório dos custos de antecipação e atraso da produção.

3. MODELAGEM MATEMÁTICA

O modelo matemático desenvolvido foi baseado nos trabalhos de [2] e [13] e pode ser descrito como se segue.

Sejam s_i a data de início do processamento do *job i* ($s_i \geq 0$), e_i o tempo de antecipação do *job i* e t_i o tempo de atraso do *job i*. Diferentemente de [2], foram utilizados dois *jobs* fictícios, 0 (zero) e n+1, de tal forma que 0 antecede imediatamente a primeira operação e n+1 sucede imediatamente o último *job* na seqüência de produção. Admite-se que P_0 e P_{n+1} são iguais a zero e que S_{0i} =0 e $S_{i,n+1}$=0, $\forall i$ = 1, ..., n.

Seja y_{ij} uma variável binária definida da seguinte forma:

$$y_{ij} = \begin{cases} 1, & \text{se o } job\ j \text{ é sequenciado imediatamente após o } job\ i; \\ 0, & \text{caso contrário.} \end{cases}$$

Considere, ainda, uma constante M de valor suficientemente grande. O modelo de PLIM proposto é apresentado a seguir:

A função objetivo, representada pela equação (3.1), tem como critério de otimização a minimização dos custos de antecipação e atraso. As restrições (3.2) definem a seqüência de operações sobre o recurso (máquina) utilizado, ou seja, elas garantem que haja um tempo suficiente para completar um *job i* antes de começar um *job j*. As restrições (3.3) e (3.4) garantem que cada *job* tenha somente um *job* imediatamente antecessor e um *job* imediatamente sucessor, respectivamente. As restrições (3.5) e (3.6) definem os valores do atraso e da antecipação de acordo com a janela de entrega desejada para o término do processamento do *job i*, caso estes existam. As restrições (3.7) a (3.10) definem o tipo das variáveis do problema.

minimizar $$Z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \qquad (3.1)$$

sujeito a:

$$s_j - s_i - y_{ij}(M + S_{ij}) \geq P_i - M \quad \forall\ i = 0,1,..., n;\ \forall\ j = 1, 2, ..., n+1 \text{ e } i \neq j \qquad (3.2)$$

$$\sum_{j=1,\, j\neq i}^{n+1} y_{ij} = 1 \quad \forall\ i = 0,1,..., n \qquad (3.3)$$

$$\sum_{i=0,\, i\neq j}^{n} y_{ij} = 1 \quad \forall\ j = 1, 2, ..., n+1 \qquad (3.4)$$

$$s_i + P_i + e_i \geq E_i \quad \forall\ i = 1, 2, ..., n \qquad (3.5)$$

$$s_i + P_i - t_i \leq T_i \quad \forall\ i = 1, 2, ..., n \qquad (3.6)$$

$$s_i \geq 0 \quad \forall\ i = 0, 1, ..., n+1 \qquad (3.7)$$

$$e_i \geq 0 \quad \forall\ i = 1, 2, ..., n \qquad (3.8)$$

$$t_i \geq 0 \quad \forall\ i = 1, 2, ..., n \qquad (3.9)$$

$$y_{ij} \in \{0,1\} \quad \forall\ i,\ j = 0, 1, ..., n+1 \qquad (3.10)$$

4. ALGORITMO DE DETERMINAÇÃO DAS DATAS ÓTIMAS DE INÍCIO DE PROCESSAMENTO DOS *JOBS*

Dada uma seqüência de *jobs*, o algoritmo de determinação das datas ótimas de início de processamento (ADDOIP) determina a melhor data para se iniciar o processamento de cada *job*, de acordo com a janela de entrega do mesmo.

O ADDOIP proposto é baseado nos trabalhos de [17] e [10]. Como a seqüência é conhecida, incorporou-se ao tempo de processamento do *job j* o tempo de preparação S_{ij}, quando o *job j* é processado imediatamente após o *job i*.

Sejam V a seqüência de *jobs* dada e C_k a data de conclusão do processamento do *job k*. Suponha que um sub-conjunto de *jobs* $B_j=\{J_0, J_1, ..., J_m\} \subseteq V$ forme um bloco no seqüenciamento, ou seja, os *jobs* em B_j são seqüenciados consecutivamente sem tempo ocioso entre eles. Além disso, assume-se que há l blocos em V e sejam $prim(j)$ e $ult(j)$ o primeiro e o último *job* no bloco B_j, respectivamente.

O processamento do primeiro *job* (J_1) da seqüência V é programado para ser finalizado na data E_{J1} se $P_{J1} \leq E_{J1}$ ou iniciado na data 0 (finalizando na data P_{J1}) se $P_{J1} > E_{J1}$.

Os demais *jobs* são programados da seguinte forma:

Se $C_{Jk} + S_{(Jk)(Jk+1)} + P_{Jk+1} < E_{Jk+1}$, então o *job* J_{k+1} tem seu processamento programado para ser finalizado na data E_{Jk+1}, isto é, o *job* J_{k+1} tem seu processamento programado para ser finalizado no início de sua janela de entrega, desta forma, iniciando um novo bloco.

Por outro lado, se $C_{Jk} + S_{(Jk)(Jk+1)} + P_{Jk+1} \geq E_{Jk+1}$, então o *job* J_{k+1} tem seu processamento programado para ser finalizado na data $C_{Jk} + S_{(Jk)(Jk+1)} + P_{Jk+1}$ e é incluído como último elemento do bloco corrente.

Após a determinação da data de conclusão do processamento de cada *job*, é necessário verificar o posicionamento dos blocos.

O custo mínimo do bloco B_j ocorre nos pontos extremos de sua função custo, isto é, no início ou no final da janela de entrega de um dos *jobs* no bloco. Por causa da natureza linear convexa por partes da função custo, o custo mínimo pode ser facilmente obtido pela comparação dos somatórios das penalidades dos *jobs* no bloco, no início e no final de cada janela de entrega no bloco (penalidades por antecipação são consideradas negativas enquanto penalidades por atraso são consideradas positivas). Estes somatórios fornecem as inclinações das retas (m) pertencentes à função custo.

O custo mínimo do bloco B_j se dá no ponto extremo onde a inclinação m torna-se maior ou igual a zero. Quando o ponto mínimo do bloco B_j é encontrado, todo o bloco é movido em direção ao ponto mínimo até que um dos três casos seguintes aconteça:

(1) $s_{prim(j)} = 0$.

(2) O ponto mínimo é alcançado.

(3) $s_{prim(j)}$ torna-se igual a $C_{ult(j-1)} + S_{(ult(j-1))(prim(j))}$.

No caso (3), o bloco B_j é unido ao bloco B_{j-1}, resultando em um novo bloco B_{j-1}. Então, o ponto mínimo do novo B_{j-1} deve ser obtido. O procedimento anterior deve ser realizado até que o caso (1) e (2) ocorram para cada bloco.

Proposição ([10]): O custo total de uma dada seqüência de *jobs* alcança seu valor ótimo se cada bloco B_j na seqüência alcançar seu ponto mínimo, com exceção de $s_{prim(1)}$ que pode ser igual a zero para o primeiro bloco.

O ADDOIP para uma seqüência de *jobs* dadas é descrito na Fig. 1, onde o procedimento *MudaBloco* movimenta o bloco B_j até seu ponto de mínimo e, caso seja necessário, combina-o com o bloco anterior.

```
procedimento Determina_Datas_Otimas_Inicio (n, v, f(v))
1    {Inicialização}
2    B ← 1;  prim(B) ← ult(B) ← 1;
```

```
3    s_1 = max{0, E_1 − P_1};
4    C_1 = max{E_1, P_1};
5    {Demais Jobs}
6    para i = 2 até n (passo=1) faça
7        se (C_{i−1} + P_i + S_{(i−1)(i)} < E_i) então
8            B ← B + 1;  prim(B) ← ult(B) ← i;
9            s_i ← E_i − P_i + S_{(i−1)(i)};  C_i ← E_i;
10       Senão
11           se (C_{i−1} + P_i + S_{(i−1)(i)} = E_i) então
12               ult(B) ← i;
13               s_i ← C_{i−1} + S_{(i−1)(i)};  C_i ← E_i;
14           Senão
15               ult(B) ← i;
16               s_i ← C_{i−1} + S_{(i−1)(i)};  C_i ← s_i + P_i;
17           fim-se;
18       fim-se;
19       repita (até que todos os blocos estejam no ponto mínimo) ou ( s_{prim(1)} = 0 )
20           MudaBloco(B);
21       fim-repita;
22   fim-para;
23   Determina f(v);
Fim Determina_Datas_Otimas_Inicio;
```

Figura 1: Algoritmo para Determinação das Datas Ótimas de Início de Processamento

Para exemplificar o método, seja a Tabela 1, relativa a 4 *jobs*, em que se apresenta o tempo de processamento (P_i), o custo por antecipação (α_i), o custo por atraso (β_i), os tempos de preparação da máquina (S_{ij}), a data inicial (E_i) e a data final (T_i) desejadas para entrega. Seja também a seqüência V={3, 4, 1, 2}.

Tabela 1: Dados do Exemplo

Jobs	**Dados do Problema**					**Tempo de Preparação da Máquina**			
	P_i	E_i	T_i	α_i	β_i	**1**	**2**	**3**	**4**
1	3	3	7	9	18	0	1	1	2
2	5	11	16	10	20	1	0	3	2
3	4	5	9	7	14	1	3	0	2
4	5	7	13	8	16	2	2	2	0

Inicialização:
Determinando as datas de início e término do processamento do primeiro *job* na seqüência:
$J_1 = 3;\ s_{J_1} = s_3 = \max\{0, E_3 - P_3\} = \max\{0, 5-4\} = 1;\ C_{J_1} = C_3 = \max\{E_3, P_3\} = E_3 = 5$.

Primeira Iteração:
Determinando as datas de início e término do processamento do segundo *job* na seqüência:
$J_2 = 4;\ C_3 + S_{34} + P_4 = 5 + 2 + 5 = 12$.

Como $C_3 + S_{34} + P_4 > E_4\ (12 > 7)$, o *job* 4 é inserido como último elemento do bloco corrente.
$s_4 = C_3 + S_{34} = 5 + 2 = 7;\ C_4 = s_4 + P_4 = 7 + 5 = 12$.

É necessário verificar neste momento a posição do bloco para verificar se ele está em seu ponto mínimo. Para isto, primeiramente, determina-se as possíveis datas de início do processamento do bloco. Essas datas são obtidas posicionando cada *job* do bloco no início e no final de sua janela de entrega, pois como foi visto o custo mínimo do bloco se dá no início ou no final da janela de entrega de um dos *jobs* no bloco. No exemplo mostrado, as possíveis datas de início do processamento do bloco são: –4, 1, 2 e 5. Determinadas as possíveis datas de início de processamento, calculam-se então as inclinações *m* de acordo com essas datas. Para calcular essas inclinações, as penalidades por antecipação assumem valores negativos e as penalidades por atraso assumem valores positivos. O ponto mínimo, o qual deseja-se

encontrar, se dá no ponto aonde a inclinação *m* se torna maior ou igual a zero. No exemplo apresentado, se o processamento do bloco iniciar antes da data –4, ou seja, com todos os *jobs* sendo antecipados, então a inclinação *m* é dada pelo somatório das penalidades por antecipação dos *jobs* 3 e 4, portanto *m* = –7 –8 = –15. Agora, se o processamento do bloco iniciar entre a data –4 e 1, o *job* 4 deixa de estar antecipado, e a inclinação *m* passa a ser *m*= –7 (penalidade por antecipação do *job* 3). Agora, se o processamento do bloco iniciar entre a data 1 e 2, o *job* 3 deixa de estar antecipado, e a inclinação *m* passa a ser *m*= 0. Como *m* tornou-se maior igual a zero este ponto é o ponto mínimo do bloco, ou seja, o custo mínimo do bloco se dá quando o processamento do bloco se inicia na data 1. Como o início do processamento bloco já é na data 1 então não é necessário movimentar o bloco até esta data. Esta situação é ilustrada na Fig. 2. Nesta figura são apresentadas, ainda, as demais inclinações, mas no momento em que a inclinação torna-se maior ou igual a zero, o procedimento é interrompido, pois este é o ponto mínimo do bloco.

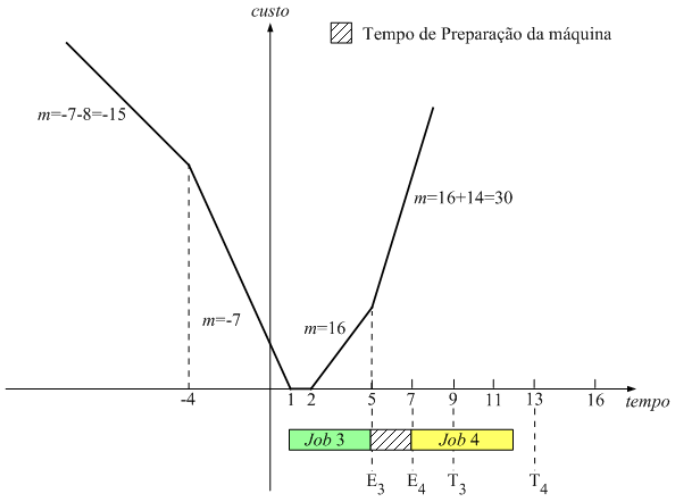

Figura 2: Situação dos *jobs* após a primeira iteração do ADDOIP

Segunda Iteração:
Determinando as datas de início e término do processamento do terceiro *job* na seqüência:
$J_3 = 1;\ C_4 + S_{41} + P_1 = 12 + 2 + 3 = 17$.
Como $C_4 + S_{41} + P_1 > E_1\,(17 > 3)$, o *job* 1 é inserido como último elemento do bloco corrente.
$s_1 = C_4 + S_{41} = 12 + 2 = 14;\ C_1 = s_1 + P_1 = 14 + 3 = 17$.

Verificando a posição do bloco:

As possíveis datas de início do processamento do bloco são: –13, –11, –4, 1, 2 e 5. As inclinações *m* calculadas de acordo com estas datas são mostradas na Fig. 3(a). Nesta figura verifica-se que na data –11 a inclinação *m* torna-se maior ou igual a zero, sendo este, então, o ponto mínimo do bloco. Como pode ser visto também nesta figura o bloco se encontra no instante 1, então o bloco deve ser movido até um dos três casos citados anteriormente acontecerem. No exemplo considerado, o bloco é empurrado até o caso (1) ocorrer, ou seja, $s_{prim(1)} = s_3 = 0$. A Fig. 3(b) apresenta o bloco já movido até o instante 0.

Este procedimento se repete até que todos os *jobs* estejam seqüenciados.

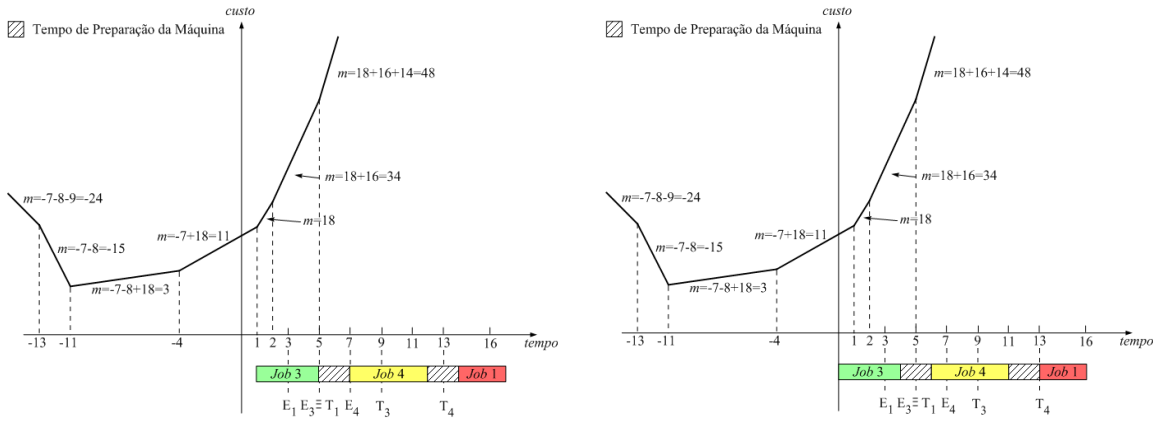

Figura 3: Situação dos *jobs* antes e depois da determinação do ponto mínimo do bloco na segunda iteração

5. ILS E GRASP APLICADOS AO PSUMAA

Nesta seção são apresentadas as principais características dos métodos heurísticos híbridos propostos para resolver o problema proposto.

5.1. REPRESENTAÇÃO DE UMA SEQÜÊNCIA

Uma solução para o problema é representada por um vetor v de n posições, com cada posição v_i indicando a ordem de produção do i-ésimo *job*.

Por exemplo, para seis *jobs*, uma possível solução é a seqüência $v = \{3, 2, 6, 4, 1, 5\}$, na qual o *job* 3 é o primeiro a ser realizado e o *job* 5, o último.

5.2. VIZINHANÇA DE UMA SOLUÇÃO

Para explorar o espaço de soluções são utilizados dois tipos de movimentos: troca da ordem de processamento de dois *jobs* da seqüência de produção e realocação de um *job* para outra posição na seqüência de produção. Esses movimentos definem, respectivamente, as estruturas de vizinhança $N^{(T)}$ e $N^{(R)}$.

Dado um conjunto com n *jobs*, na primeira estrutura de vizinhança, $N^{(T)}$, há $n(n\text{-}1)/2$ movimentos possíveis, ou equivalentemente, esse mesmo número de vizinhos. Por exemplo, a solução $v' = \{4, 2, 6, 3, 1, 5\}$ é um vizinho da solução v, pois é obtido deste pela troca do primeiro com o quarto *job*. Já na segunda estrutura de vizinhança, $N^{(R)}$, há $(n\text{-}1)^2$ movimentos possíveis. Por exemplo, a solução $v' = \{2, 6, 4, 3, 1, 5\}$ é obtida de v pela realocação do *job* da primeira posição para a quarta posição na seqüência de produção. Nessa estrutura são permitidos movimentos para posições sucessoras ou antecessoras à posição em que o *job* se encontra na seqüência de produção.

5.3. FUNÇÃO DE AVALIAÇÃO

Como os movimentos propostos não produzem soluções inviáveis, uma solução do problema é avaliada pela própria função objetivo, dada pela expressão (3.1) do modelo de PLIM.

5.4. GERAÇÃO DA SOLUÇÃO INICIAL

O método proposto para construir uma solução inicial para o PSUMAA baseia-se na fase de construção da metaheurística GRASP, onde os elementos são inseridos um a um na solução de acordo com um critério de seleção, o qual estima o benefício da seleção de cada elemento (*job*). A cada iteração do método de construção GRASP, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista de candidatos, seguindo um critério de ordenação pré-determinado. Os melhores candidatos, de acordo com um o critério de seleção, são colocados em uma lista restrita de candidatos (LRC). A seguir, um desses candidatos é escolhido aleatoriamente e inserido na solução corrente. Um parâmetro γ,

com $0 \leq \gamma \leq 1$, é utilizado para controlar o tamanho da LRC. Para estimar o benefício da inserção de cada *job*, utiliza-se a data de início de entrega desejada como critério de seleção. O procedimento proposto é, portanto, uma variante da heurística construtiva EDD (*Earliest Due Date*), a qual seqüencia os *jobs* em ordem crescente da data de início desejada para a entrega do *job*.

Na Figura 4 apresenta-se o pseudocódigo da fase de construção GRASP aplicada ao problema de seqüenciamento em questão. Nesta figura, E_{min} representa cronologicamente a primeira data de início de entrega e E_{max}, a última.

```
procedimento Constroi_Solucao_GRASP (n, v, γ)
1    v ← φ ;
2    Inicialize o conjunto LC de jobs candidatos;
3    enquanto ( LC ≠ φ ) faça
4         E_min = min_{i∈LC}{E_i} ;
5         E_max = max_{i∈LC}{E_i} ;
6         LRC = {i ∈ LC | E_i ≤ E_min + γ(E_max − E_min)} ;
7         Selecione, aleatoriamente, um elemento i ∈ LRC;
8         v ← v ∪ {i} ;
9         Atualize o conjunto LC de jobs candidatos;
10   Fim-enquanto;
11   Retorne v;
fim Constroi_Solucao_GRASP;
```

Figura 4: Algoritmo para Construir uma Solução via GRASP

Para gerar a solução inicial, o método proposto aplica a fase de construção GRASP por *MaxIter* vezes e retorna a melhor solução construída. O método inicia com uma solução gulosa (a qual é obtida fazendo-se $\gamma = 0$ na primeira aplicação da fase de construção GRASP) e nas demais iterações é selecionado, aleatoriamente, um valor para γ dentro de um conjunto de valores Γ previamente definidos. Após *MaxIter* iterações, retorna-se a melhor solução encontrada. Este método de geração de uma solução inicial é esquematizado na Figura 5.

```
procedimento Constroi_Solucao_Inicial (n, v, MaxIter)
1    Constroi_Solucao_GRASP(n, v', γ=0); {Constrói uma solução gulosa}
2    Determina_Datas_Otimas_Inicio(n, v', f(v'));
3    v* ← v' ;
4    f* ← f(v') ;
5    para i = 1 até MaxIter (passo = 1) faça
6         Selecione, aleatoriamente, um valor γ ∈ Γ;
7         Constroi_Solucao_GRASP(n, v', γ);
8         Determina_Datas_Otimas_Inicio(n, v', f(v'));
9         Se ( f(v') < f* ) então
10             v* ← v' ;   f* ← f(v') ;
11        fim-se;
12   fim-para;
13   Retorne v*;
Fim Constroi_Solucao_Inicial;
```

Figura 5: Algoritmo para Gerar uma Solução Inicial

5.5. MÉTODO GRASP-VND-RT

O primeiro método, denominado GRASP-VND-RT, combina características dos procedimentos GRASP e VND (*Variable Neighborhood Descent*), conforme a seguir se descreve.

O método GRASP é composto de duas fases: (*i*) fase de construção e (*ii*) fase de melhoramento (busca local). Na fase de construção, uma solução é gerada, elemento a elemento, de forma parcialmente gulosa e na fase de melhoramento é aplicado um método de

busca local à solução gerada na primeira fase. O método é repetido por um certo número de iterações. A melhor solução encontrada ao longo de todas as iterações do método é retornada como resultado.

No método GRASP proposto, a solução inicial é gerada de acordo com a metodologia descrita na Seção 5.4, Figura 3. A busca local é feita pelo Método de Descida em Vizinhança Variável (VND – *Variable Neighborhood Descent*). O VND é um método de refinamento que consiste em explorar o espaço de soluções através de trocas sistemáticas de estruturas de vizinhança, aceitando somente soluções de melhora da solução corrente e retornando à primeira estrutura quando uma solução melhor é encontrada.

No método GRASP-VND-RT, a busca local feita pelo VND primeiramente usa o Método Randômico de Descida (MRD) com a estrutura de vizinhança $N^{(R)}$. Em seguida, usa o MRD com a vizinhança $N^{(T)}$. Sempre que há uma melhora na solução, o método VND retorna à estrutura de vizinhança anterior. O MRD gera um vizinho v' aleatoriamente de acordo com uma das estruturas de vizinhança adotadas ($N^{(R)}(v)$ ou $N^{(T)}(v)$) e é executado até que o número de iterações sem melhora seja maior ou igual ao número máximo de iterações sem melhora (*IterMax*2).

Como o MRD não garante que a solução resultante pela aplicação do método seja um ótimo local com relação às vizinhanças consideradas, aplicou-se ao final do método o procedimento VND, adotando-se como método de busca local o Método da Descida (MD), tendo $N^{(R)}$ como primeira estrutura de vizinhança e $N^{(T)}$ como segunda estrutura de vizinhança. Isto é, dada uma solução *v*, toda sua vizinhança baseada em movimentos de realocação é analisada, movendo-se apenas para o vizinho que produzir melhora. Não havendo vizinhos de melhora, passa-se para a vizinhança seguinte, que opera com movimentos de troca. Havendo melhora, volta-se à vizinhança de realocação. O método termina quando não há melhora nem com movimentos de realocação nem com movimentos de troca de *jobs*.

O pseudocódigo do método GRASP-VND-RT é esquematizado na Figura 6.

```
procedimento GRASP-VND-RT
1    v0 ← Constroi_Solucao_Inicial (n, v0, MaxIter);
2    v ← MetodoBuscaLocal (n, v0);
3    f* ← f(v); Iter ← 0;
4    enquanto (Iter < IterMax) faça
5        v' ← Constroi_Solucao_Inicial (n, v0, MaxIter);
6        v' ← MetodoBuscaLocal (n, v');
7        se (f(v') < f*) então
8            v ← v';
9            f* ← f(v');
10       fim-se;
11       Iter ← Iter + 1;
12   fim-enquanto;
13   v ← VND_MD(v);
14   Retorne v;
fim GRASP-VND-RT;
```

Figura 6: Método GRASP-VND-RT proposto

5.6. MÉTODO GILS-VND-RT

O segundo método proposto, denominado GILS-VND-RT, combina características dos procedimentos GRASP, ILS (*Iterated Local Search*) e VND (*Variable Neighborhood Descent*), conforme a seguir se descreve.

O método ILS (*Iterated Local Search*) é baseado na idéia de que um procedimento de busca local pode ser melhorado gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações na solução ótima local. A perturbação precisa ser suficientemente forte para permitir que a busca local explore diferentes soluções, mas também fraca o

suficiente para evitar um reinício aleatório. Para aplicar um algoritmo ILS, quatro componentes têm que ser especificadas: (*i*) um procedimento que gera uma solução inicial v_0 para o problema; (*ii*) um procedimento de busca local, que retorna uma solução melhorada v''; (*iii*) um procedimento de perturbação, que modifica a solução corrente v guiando a uma solução intermediária v' e (*iv*) um procedimento que decide de qual solução a próxima perturbação será aplicada.

No método GILS-VND-RT, a geração da solução inicial e o método de busca local utilizados são os mesmos utilizados pelo método GRASP-VND-RT.

As perturbações são feitas por meio de trocas aleatórias entre dois *jobs* quaisquer da solução. Caso o nível de perturbação seja igual a *Nperturb*, então são feitas *Nperturb*+1 trocas aleatórias. O nível de perturbação é aumentado em uma unidade quando a busca local não resultar em melhoria da solução corrente e quando no mínimo 10% da vizinhança desta solução tiver sido explorada. Quando há melhora, o nível de perturbação volta para seu nível mínimo, isto é, 1, voltando a se repetir o processo.

No método proposto move-se de uma solução para outra somente se o ótimo local encontrado for melhor que o ótimo local corrente, ou seja, se $f(v'') < f(v)$. O método pára após um número máximo de iterações sem melhora no valor da melhor solução encontrada até então (*IterMax*1). O pseudocódigo do método GILS-VND-RT é esquematizado na Figura 7.

Assim como no método GRASP-VND-RT, aplicou-se ao final do método o procedimento VND, adotando-se como método de busca local o Método da Descida (MD), tendo $N^{(R)}$ como primeira estrutura de vizinhança e $N^{(T)}$ como segunda estrutura de vizinhança.

```
procedimento GILS-VND-RT
1    v0 ← Constroi_Solucao_Inicial (n, v0 , MaxIter);
2    v ← MetodoBuscaLocal (n, v0 );
3    nivel ← 1; Iter ← 0;
4    enquanto (Iter < IterMax2) faça
5        v' ← Perturbacao(nivel, v) ;
6        v'' ← VND _ MRD(n, v', IterMax1) ;
7        se (f(v'') < f(v)) então
8            v ← v'' ;
9            nivel ← 1;   Iter ← 0 ;
10       Senão
11           se (pelo menos 10% da vizinhança no nível corrente foi explorada)
12               então nivel ← nivel + 1;
13           fim-se;
14           Iter ← Iter + 1;
15       fim-se;
16   fim-enquanto;
17   v ← VND _ MD(v) ;
18   Retorne v;
fim GILS-VND-RT;
```

Figura 7: Método GILS-VND-RT proposto

6. RESULTADOS E ANÁLISE

Os testes computacionais foram feitos em um computador Atlon XP 64 Bits 3000+ (aproximadamente 2 GHz), com 1 GB de memória RAM, sob plataforma Windows XP.

Os problemas-teste utilizados foram gerados usando métodos pseudo-aleatórios baseados nos trabalhos de [17], [12] e [14], com número de *jobs n* igual a 8, 9, 10, 11, 12, 15, 20, 25, 30, 40, 50 e 75.

Os tempos de processamento P_i dos *jobs* são uniformemente e discretamente distribuídos entre 1 e 100. Os centros das janelas de entrega são uniformemente e discretamente distribuídos dentro do intervalo [(1-*TF*-*RDD*/2)×*MS*, (1-*TF*+*RDD*/2)×MS],

onde *MS* é o tempo total de processamento de todos os *jobs*, *TF* é o fator de atraso e *RDD* é a variação relativa da janela de entrega. Os valores considerados para *TF* foram 0,1; 0,2; 0,3 e 0,4 e os valores considerados para *RDD* foram 0,8; 1,0 e 1,2. Os tamanhos das janelas de entrega são uniformemente e discretamente distribuídas no intervalo [1, *MS*/*n*].

Os custos por atraso da produção (β_i) são uniformemente e discretamente distribuídos dentro do intervalo [20, 100]. Na maioria dos casos reais, o atraso da produção é menos desejável do que a sua antecipação. Desta forma, os custos por antecipação da produção (α_i) são gerados como *k* vezes o custo por atraso do mesmo *job*, sendo *k* uma variável aleatória uniforme entre 0 e 1. Os tempos de preparação da máquina (S_{ij}) são gerados uniformemente e discretamente dentro do intervalo [0, 50]. Utilizaram-se tempos de preparação simétricos, ou seja, $S_{ij} = S_{ji}$.

Como *TF* possui quatro valores diferentes e *RDD* três valores diferentes, foram gerados 12 problemas-teste para cada número de *jobs*, totalizando 144 problemas-teste.

O modelo matemático apresentado na Seção 3 foi implementado usando a ferramenta de modelagem AMPL e resolvido pelo *software* de otimização CPLEX, versão 9.1 da ILOG. Foram resolvidos os problemas-teste de 8 a 12 *jobs*, sendo os resultados apresentados na Tabela 2. Nesta tabela a primeira coluna apresenta o número de *jobs* do conjunto de problemas-teste, a segunda coluna apresenta o tempo computacional médio gasto para se chegar à solução ótima e a última coluna apresenta o percentual de soluções ótimas encontradas dentro dos 12 problemas-teste para cada número de *jobs*. Considerou-se um tempo limite de 3600 segundos para obtenção de uma solução ótima. Como pode ser observado na Tabela 2, o modelo de programação matemático se mostra eficiente para resolver problemas-teste com até 10 *jobs*. Para os problemas de 11 e 12 *jobs*, o otimizador não foi capaz de encontrar ótimos em todos os casos. Acima de 12 *jobs* o CPLEX não foi capaz de encontrar soluções ótimas dentro desse tempo limite.

Tabela 2: Resultados do Modelo Matemático

Número de *Jobs*	Tempo de processamento Médio (s)	Percentual de Soluções Ótimas Encontradas (%)
8	2,22	100
9	43,38	100
10	93,39	100
11	1931,25	75
12	2022,72	50

Os métodos propostos foram implementados em linguagem C, utilizando o ambiente C++ Builder 5. Os parâmetros dos métodos, obtidos experimentalmente, são apresentados na Tabela 3.

Cada problema-teste foi resolvido 30 vezes pelo método proposto, com exceção daqueles envolvendo 75 *jobs*, os quais foram resolvidos apenas 20 vezes, pois demandavam um tempo computacional mais elevado. Para os problemas-teste de 75 *jobs* adotou-se, ainda, o valor $2 \times n$ como número máximo de iterações sem melhora do MRD utilizado pelo VND para reduzir o tempo de processamento demandado.

Tabela 3: Parâmetros dos Métodos

Parâmetros	Valores
Conjunto de valores (Γ) para o parâmetro γ da fase de construção	0,02; 0,04; 0,12; 0,14
Nro máximo de iterações do procedimento de Geração da Solução Inicial (*MaxIter*)	52
Número máximo de iterações do método GRASP (*IterMax*)	$2 \times n$
Número máximo de iterações sem melhora do ILS (*IterMax*1)	$1{,}4 \times n$
Número máximo de iterações sem melhora do MRD (*IterMax*2)	$7 \times n$

A Tabela 4 apresenta os resultados obtidos para os métodos propostos. Nesta tabela, os resultados obtidos com os problemas-teste de 8 a 12 *jobs* são comparados com os resultados ótimos obtidos pelo modelo matemático. Para os demais problemas-teste, os valores foram comparados com o melhor resultado obtido pela aplicação dos procedimentos

heurísticos considerando todas as execuções. O GAP é calculado pela expressão *GAP* = (*RMed* - *MR*) / *MR* × *100*, onde *RMed* é o resultado médio obtido pela aplicação do método e *MR* é o melhor resultado obtido para cada problema. O GAP da melhor solução é obtido pela aplicação da fórmula anterior substituindo-se o resultado médio (*RMed*) pelo melhor resultado obtido pela aplicação do método.

Pela Tabela 4, observa-se que para os problemas de 8 a 12 *jobs*, os desvios da solução média produzida por ambos os métodos foram baixos (inferiores a 0,66%). Ainda para os problemas de 8 a 12 *jobs*, os métodos utilizados quase sempre alcançaram o resultado ótimo conhecido. Para problemas de dimensões superiores a 12 *jobs*, a média dos GAPs da solução média do método GRASP-VND-RT variou de 1,23% a 13,69% e no método GILS-VND-RT variou 1,47% a 10,89%. Os tempos de processamento médio do método GRASP-VND-RT variaram de 1,12 a 1856,27 segundos. Já no GILS-VND-RT, os tempos de processamento médio variaram de 0,94 a 1368,08 segundos. Para os problemas-teste com 12 ou menos *jobs*, nota-se que o tempo gasto por ambos os métodos propostos foi sempre inferior àquele demandado pelo otimizador aplicado ao modelo de PLIM. Nota-se, ainda, que para o problema com 12 *jobs*, limite para utilização do modelo matemático, demandou-se apenas 0,38 segundos pelo método GRASP-VND-RT e 0,29 segundos pelo método GILS-VND-RT contra 2023 segundos, em média, do modelo de PLIM. Observa-se, também, que o método GILS-VND-RT teve uma pequena vantagem sobre o método GRASP-VND-RT, tanto no desvio médio (2,52%) quanto no desvio da melhor solução (0,07%).

Tabela 4: Resultados dos Métodos GRASP-VND-RT e GILS-VND-RT

Número de *Jobs*	GRASP-VND-RT			GILS-VND-RT		
	MGSM[(1)] (%)	MGMS[(2)] (%)	TPM[(3)] (s)	MGSM[(1)] (%)	MGMS[(2)] (%)	TPM[(3)] (s)
8	0,04	0,00	0,06	0,03	0,00	0,04
9	0,66	0,00	0,10	0,06	0,00	0,07
10	0,37	0,00	0,17	0,02	0,00	0,11
11	0,55	0,00	0,26	0,12	0,00	0,20
12	0,29	0,27	0,38	0,21	0,00	0,29
15	1,23	0,00	1,12	1,47	0,00	0,94
20	2,24	0,00	4,55	1,65	0,00	4,35
25	3,17	0,00	13,48	2,32	0,00	13,29
30	5,05	0,59	39,08	3,34	0,19	40,07
40	7,12	1,19	156,98	4,18	0,00	155,79
50	10,42	1,67	496,21	5,89	0,04	492,28
75	13,69	1,47	1856,27	10,89	0,56	1368,08
Média	**3,74**	**0,43**	**214,06**	**2,52**	**0,07**	**172,96**

[(1)] Média dos GAPs da Solução Média; [(2)] Média dos GAPs da Melhor Solução; [(3)] Tempo de processamento Médio, em segundos.

7. CONCLUSÕES

Este artigo teve seu foco no Problema de Seqüenciamento em uma Máquina com penalidades por Antecipação e Atraso da produção (PSUMAA), considerando janelas de entrega e tempo de preparação da máquina dependente da seqüência de produção. Propôs-se inicialmente um modelo de PLIM para representar o problema estudado, sendo este resolvido pelo CPLEX, versão 9.1.

Em seguida, foram também propostos dois métodos heurísticos inéditos baseados em GRASP, *Iterated Local Search* e *Variable Neighborhood Descent* para resolvê-lo. A resolução heurística envolve dois passos na geração de uma solução: (*i*) determinação da seqüência dos *jobs* e (*ii*) determinação da data ótima de início do processamento de cada *job* da seqüência.

Os métodos heurísticos propostos foram aplicados em problemas-teste gerados por procedimentos pseudoaleatórios e mostraram-se eficientes para resolver o PSUMAA. Em problemas de pequenas dimensões (8 a 12 *jobs*), ambos os métodos produziram soluções de alta qualidade, tendo quase sempre alcançado a solução ótima e os valores médios das

soluções finais foram bastante próximos aos valores das soluções ótimas. Os tempos de processamento para esses problemas foram bem menores do que aqueles despendidos com a resolução do modelo matemático. Em problemas de dimensões maiores (15 a 75 *jobs*), os métodos apresentaram baixos desvios em relação à melhor solução e o tempo de processamento foi pequeno se comparado ao horizonte de planejamento (tipicamente uma semana).

8. AGRADECIMENTOS

O primeiro autor agradece a CAPES pela bolsa concedida e ao Departamento de Produção da UFMG pela oportunidade concedida. Os dois últimos autores agradecem à FAPEMIG e à Universidade Federal de Ouro Preto pelo apoio concedido, processos TEC 679/06 e PROBIC/UFOP.

9. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Baker, K. R., Scudder, G. D.. Sequencing with Earliness and Tardiness Penalties: A Review. **Operations Research**, v. 38, p. 22–36, 1990.

[2] Bustamante, L. M.. **Minimização do Custo de Antecipação e Atraso para o Problema de Seqüenciamento de uma Máquina com Tempo de Preparação Dependente da Seqüência: Aplicação em uma Usina Siderúrgica.** Dissertação de mestrado, Programa de Pós-Graduação em Engenharia de Produção, UFMG, Belo Horizonte, 2006.

[3] Chang, P. C.. A Branch and Bound Approach for Single Machine Scheduling with Earliness and Tardiness Penalties. **Computers & Mathematics with Applications**, v. 37, p. 133–144, 1999.

[4] Feldmann, M., Biskup, D.. Single-Machine Scheduling for Minimizing Earliness and Tardiness Penalties by Meta-heuristic Approaches. **Computers & Industrial Engineering**, v. 44, p. 307–323, 2003.

[5] Feo, T.A., Resende, M.G.C.. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, v. 6, p. 109–133, 1995.

[6] Glover, F.. Future paths for Integer Programming and links to Artificial Intelligence. **Computers and Operations Research**, v. 5, p. 553–549,1986.

[7] Gupta, S. R., Smith, J. S.. Algorithms for single machine total tardiness scheduling with sequence dependent setups. **European Journal of Operational Research**, v. 75, p. 722–739, 2006.

[8] Hansen, P.. **The steepest ascent mildest descent heuristic for combinatorial programming.** *In:* Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy, 1986.

[9] Hino, C. M., Ronconi, D.P., Mendes, A. B.. Minimizing Earliness and Tardiness Penalties in a Single-Machine Problem with a Common Due Date. **European Journal of Operational Research**, v. 160, p. 190–201, 2005.

[10] Lee, C. Y., Choi, J. Y.. A Genetic Algorithm for Job Sequencing Problems with Distinct Due Dates and General Early-Tardy Penalty Weights. **Computers and Operations Research**, v. 22, p. 857–869, 1995.

[11] Li, G.. Single Machine Earliness and Tardiness Scheduling. **European Journal of Operational Research**, v. 96, p. 546–558, 1997.

[12] Liaw, C.F.. A Branch-and-Bound Algorithm for the Single Machine Earliness and Tardiness Scheduling Problem. **Computers & Operations Research**, v. 26, p. 679–693, 1999.

[13] Manne, A. S.. On the Job-shop Scheduling Problem. **Operations Research**, v. 8, p. 219–223, 1960.

[14] Mazzini, R., Armentano, V. A.. A Heuristic for Single Machine Scheduling with Early and Tardy Costs. **European Journal of Operational Research**, v. 128, p. 129–146, 2001.

[15] Mladenovic, N., Hansen, P.. Variable Neighborhood Search. **Computers and Operations Research**, v. 24, p. 1097–1100, 1997.

[16] Rabadi, G., Mollaghasemi, M., Anagnostopoulos, G. C.. A Branch-and-Bound Algorithm for the Early/Tardy Machine Scheduling Problem with a Common due-date and Sequence-Dependent Setup Time. **Computers & Operations Research**, v. 31, p. 1727–1751, 2004.

[17] Wan, G. e Yen, B. P. C.. Tabu Search for Single Machine Scheduling with Distinct Due Windows and Weighted Earliness/Tardiness Penalties. **European Journal of Operational Research**, v. 142, p. 129–146, 2002.