ALOCAÇÃO DE TEMPOS OCIOSOS EM UMA DADA SEQUÊNCIA DE PRODUÇÃO COM JANELAS DE ENTREGA

Bruno Ferreira Rosa, Sérgio Ricardo de Souza
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Av. Amazonas, 7675, CEP 30510-000, Belo Horizonte (MG), Brasil
brunorosa@div.cefetmg.br, sergio@dppg.cefetmg.br

Marcone Jamilson Freitas Souza
Departamento de Computação - Universidade Federal de Ouro Preto (UFOP)
Campus Universitário, Morro do Cruzeiro, CEP 35.400-000, Ouro Preto (MG), Brasil
marcone@iceb.ufop.br

RESUMO

Este trabalho aborda o problema de sequenciamento de tarefas em uma máquina. O objetivo é minimizar as penalidades por antecipação e atraso da produção, considerando janelas de entrega distintas. O problema em questão pode ser dividido em dois subproblemas, a saber, determinar a sequência de execução das tarefas e decidir o momento em que cada tarefa será executada nessa sequência. É proposto um algoritmo de alocação ótima de tempos ociosos em uma dada sequência de execução das tarefas. Propõe-se, também, um algoritmo de enumeração implícita (EI) e um algoritmo *General Variable Neighborhood Search* (GVNS) que utilizam o algoritmo de alocação de tempos ociosos para resolver o problema. Experimentos computacionais mostram que o algoritmo de alocação de tempos ociosos proposto é mais eficiente que o algoritmo até então utilizado na literatura, enquanto os algoritmos EI e GVNS se mostraram boas opções para resolver o problema.

PALAVRAS CHAVE. Sequenciamento em uma máquina. Janelas de entrega. Antecipação e atraso.

Área Principal: Metaheurísticas (MH) e Otimização Combinatória (OC)

ABSTRACT

This paper addresses the single-machine job scheduling problem. The objective is to minimize the earliness and tardiness penalties, considering distinct due windows. The problem in question can be divided into two subproblems, namely, finding a job sequence and inserting optimal idle times in this sequence. An algorithm for inserting optimal idle times into a given job sequence is proposed. It is also proposed an implicit enumeration algorithm (IE) and a General Variable Neighborhood Search algorithm (GVNS) which use the idle time insertion algorithm to solve the problem. Computational experiments show that the proposed algorithm for inserting optimal idle times is more efficient than the algorithm previously used in the literature, while the EI and GVNS algorithms proved to be good options for solving the problem.

KEYWORDS. Single-machine Scheduling. Time windows. Earliness and tardiness.

Main Area: Metaheuristics (MH) and Combinatorial Optimization (OC)

1. Introdução

O problema de sequenciamento de tarefas em uma máquina com penalidades por antecipação e atraso da produção consiste em sequenciar e determinar o momento em que as tarefas devem ser executadas em uma máquina, com o objetivo de minimizar a soma ponderada das antecipações e dos atrasos na execução de tais tarefas. Reduzir as antecipações e os atrasos nas execuções das tarefas de uma atividade produtiva pode proporcionar uma significativa redução de custos. De acordo com Liaw (1999), concluir uma tarefa com atraso pode resultar em multas contratuais, perda de credibilidade da empresa e redução de vendas. Do mesmo modo, concluir uma tarefa antecipadamente pode resultar em custos financeiros extras pela necessidade de disponibilização antecipada de capital, necessidade de espaço para armazenamento ou necessidade de outros recursos para manter e gerenciar o estoque (França Filho, 2007). Este é um problema NP-difícil (Allahverdi et al., 1999) e com muitas aplicações práticas em industrias metalúrgicas, têxteis, de tintas, entre outras.

A situação mais comum nesse problema é considerar para cada tarefa uma data de entrega. No entanto, neste trabalho consideramos a existência de uma janela de entrega para cada tarefa. Segundo Wan e Yen (2002), esse caso ocorre quando existem tolerâncias em torno das datas desejadas para a conclusão de cada tarefa. Tais tolerâncias estão relacionadas às características individuais das tarefas, que influenciam nos tamanhos das suas respectivas janelas de entrega. Deste modo, as tarefas concluídas dentro de suas respectivas janelas de entrega não incorrem em nenhuma penalidade. Já aquelas concluídas fora de suas respectivas janelas de entrega geram penalidades.

Koulamas (1996) estuda este problema, doravante denotado por PSUMAA, com uma sensível distinção dos demais trabalhos da literatura. Ao contrário da literatura, que considera a existência de antecipação quando uma tarefa é concluída antes do início de tal janela, esse autor considera que há antecipação de uma tarefa quando seu processamento é iniciado antes do início de sua janela de entrega. As penalidades por antecipação e atraso são consideradas iguais para cada tarefa. O autor divide o problema em dois subproblemas, sendo um deles determinar a sequência de execução das tarefas e o outro determinar o instante ótimo de conclusão de cada tarefa na sequência dada (ou, equivalentemente, inserir tempos ociosos entre as tarefas da sequência). Para resolver o problema de determinação dos instantes ótimos de processamento de cada tarefa numa dada sequência é desenvolvido um algoritmo que insere tempos ociosos, de modo ótimo, na sequência de execução. Já o problema de sequenciamento é resolvido com adaptações de heurísticas utilizadas anteriormente em casos particulares do PSUMAA.

Wan e Yen (2002) tratam o PSUMAA tal como neste trabalho. São estudadas várias propriedades do problema a fim de facilitar sua resolução e, assim como Koulamas (1996), os autores dividem o problema em dois subproblemas. É desenvolvido um procedimento de complexidade polinomial para determinar a data ótima de conclusão de cada tarefa em uma dada sequência de execução, sendo tal procedimento uma extensão do algoritmo de Lee e Choi (1995). Por fim, uma Busca Tabu (Glover e Laguna, 1997), que faz uso do procedimento de datas ótimas e das propriedades estudadas, é utilizada para resolver o problema.

Gomes Júnior et al. (2007) focam no PSUMAA com tempos de preparação da máquina dependentes da sequência de execução. Os autores propõem um modelo de programação linear inteira mista para representar o problema, bem como um algoritmo heurístico baseado em GRASP (Feo e Resende, 1995), *Iterated Local Search* (Lourenço et al., 2003) e *Variable Neighborhood Descent* (Mladenović e Hansen, 1997) para resolvê-lo. Para cada sequência de tarefas gerada pelo algoritmo aplica-se um algoritmo de tempo polinomial para determinar a data ótima de início de processamento das tarefas na sequência dada. Tal algoritmo, denominado ADDOIP, é baseado nos trabalhos de Wan e Yen (2002) e Lee e Choi (1995).

São encontrados na literatura muitos trabalhos que fazem uso do ADDOIP em algoritmos heurísticos para resolver o PSUMAA. Entre eles, citamos: Souza et al. (2008), Rosa e Souza (2009), Rosa et al. (2010), Gonçalves e Souza (2010), Ribeiro et al. (2010) e Penna et al. (2012).

Neste trabalho é proposto um algoritmo de alocação ótima de tempos ociosos em uma

dada sequência de execução das tarefas. Propõe-se, também, um algoritmo de enumeração implícita e um algoritmo *General Variable Neighborhood Search* - GVNS (Mladenović e Hansen, 1997) para resolver o problema. Cada sequência gerada por esses algoritmos é avaliada pelo algoritmo de alocação de tempos ociosos proposto. Experimentos computacionais realizados em problemas-teste comparam a eficiência do algoritmo de alocação de tempos ociosos proposto com a do ADDOIP de Gomes Júnior et al. (2007), bem como a eficiência do algoritmo de enumeração implícita proposto com a do modelo de programação linear apresentado por esses mesmos autores.

O restante deste trabalho está organizado da seguinte forma. Na Seção 2 é feita uma descrição detalhada do PSUMAA. O algoritmo de alocação ótima de tempos ociosos proposto é apresentado na Seção 3. O algoritmo de enumeração implícita e o algoritmo GVNS são detalhados na Seção 4 e Seção 5, respectivamente. Na Seção 6 são apresentados e discutidos os resultados computacionais obtidos, enquanto a Seção 7 conclui o trabalho.

2. Características do problema abordado

O problema de sequenciamento de tarefas em uma máquina tratado neste trabalho (PSUMAA) possui as seguintes características: (*i*) Uma máquina deve processar um conjunto I de n tarefas; (*ii*) A cada tarefa $i \in I$ está associado um tempo de processamento P_i; uma janela de entrega $[E_i, T_i]$, dentro da qual a tarefa i deve ser preferencialmente concluída; um custo α_i por unidade de tempo de antecipação; e um custo β_i por unidade de tempo de atraso. Há antecipação de uma tarefa $i \in I$ quando seu processamento é concluído antes de E_i e há atraso quando seu processamento é concluído depois de T_i. Se C_i representa o instante de conclusão da tarefa i, os tempos de antecipação e atraso de i são dados por $e_i = \max(0,\ E_i - C_i)$ e $t_i = \max(0,\ C_i - T_i)$, respectivamente; (*iii*) As tarefas que forem concluídas dentro de suas respectivas janelas de entrega não geram custo; (*iv*) A máquina executa no máximo uma tarefa por vez e, uma vez iniciado o processamento de uma tarefa, não é permitida a sua interrupção; (*v*) É permitido tempo ocioso entre a execução de duas tarefas consecutivas; (*vi*) Todas as tarefas estão disponíveis na data 0.

O objetivo é determinar a sequência de execução e os instantes de início de processamento das tarefas que minimizam a soma ponderada das antecipações e atrasos de tais tarefas, ou seja, minimizar o valor de Z na equação (1):

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

3. Alocação de tempos ociosos

Resolver o PSUMAA consiste em determinar a sequência na qual as tarefas são executadas, bem como o instante de início de processamento de cada tarefa nessa sequência. Os trabalhos encontrados na literatura tratam o problema com procedimentos heurísticos que, iterativamente, geram uma sequência de execução por meio de regras de prioridade ou procedimentos de busca. Para avaliar cada sequência gerada é necessário determinar os instantes ótimos de início de processamento de cada tarefa, considerando a inserção de tempos ociosos na sequência de execução. Sendo assim, é fundamental o uso de um algoritmo que faça tal inserção de tempos o mais rápido possível.

A seguir é apresentado um algoritmo de alocação ótima de tempos ociosos em uma dada sequência de execução, denominado AAOTO. Este algoritmo foi motivado pelo trabalho de França Filho (2007), que trata um problema diferente, mas com algumas características em comum. Esse autor trata o sequenciamento de tarefas em máquinas paralelas não relacionadas, com tempos de preparação da máquina dependentes da sequência de execução e com penalidades por antecipação e atraso da produção, instantes de liberação e datas de entrega específicos para cada tarefa.

O algoritmo AAOTO proposto é constituído de duas etapas. Primeiramente, todas as tarefas são programadas o mais cedo possível, sem tempos ociosos, minimizando assim os custos de atraso e acarretando no maior custo por antecipação possível. Em seguida, tempos ociosos são inseridos de modo que custos por antecipação sejam eliminados, desde que os possíveis custos consequentemente gerados por atraso não sejam maiores.

Apesar deste trabalho tratar o PSUMAA com tempos de *setup* independentes da sequência de execução, o $AAOTO$ pode ser utilizado também para o caso em que os tempos de *setup* são dependentes da sequência de execução, tal como o ADDOIP de Gomes Júnior et al. (2007). Para considerar tal situação, seja S_{ij} o tempo de preparação da máquina entre a execução de duas tarefas consecutivas i e $j \in I$. Será assumido, também, que o tempo de preparação da máquina para o processamento da primeira tarefa na sequência é 0.

Seja $X = (x_1, x_2, \ldots, x_n)$ uma dada sequência do conjunto de tarefas I. Logo $x_i \in I$ e $x_i \neq x_j$ para todo $i,\ j \in \{1, 2, \cdots,\ n\}$. Diz-se que uma sub-sequência de tarefas $B = (x_u, x_{u+1}, \ldots, x_v) \subseteq X$, com $u \leq v$, forma um bloco na sequência X se as tarefas em B são sequenciadas consecutivamente sem tempo ocioso entre elas e existe ociosidade entre as tarefas x_{u-1} e x_u e as tarefas x_v e x_{v+1}. Para o caso em que $u = 1$, desconsidera-se a condição de existência de ociosidade entre as tarefas x_{u-1} e x_u. Analogamente, para o caso em que $v = n$, desconsidera-se a condição de existência de ociosidade entra as tarefas x_v e x_{v+1}.

Seja C_i o instante de conclusão da tarefa i. Na primeira etapa do algoritmo, o processamento da primeira tarefa (x_1) da sequência X é programado para ser iniciado na data 0, ou seja, $C_{x_1} = P_{x_1}$. O instante de conclusão das demais tarefas é dado por $C_{x_i} = C_{x_{i-1}} + S_{(x_{i-1})(x_i)} + P_{x_i}$, para $i = 2, 3, \ldots, n$.

Ao contrário do algoritmo de França Filho (2007), a segunda etapa do AAOTO começa a partir da última tarefa da sequência (x_n). Sucessivamente, da última para a primeira, verifica-se se a inserção de tempos ociosos é vantajosa. Se ao final da primeira etapa a última tarefa não estiver adiantada, então ela não sofrerá nenhum deslocamento para a direita, uma vez que isso não reduziria o custo por antecipação associado a ela. Caso contrário, ou seja, se x_n estiver antecipada, ela sofre um deslocamento φ para a direita, sendo φ dado pela equação (2).

$$\varphi = E_{x_n} - C_{x_n} \tag{2}$$

A inserção de tempos ociosos antes das demais tarefas x_i, com $i = n-1, n-2, \ldots, 1$, é realizada da seguinte forma:

Se $C_{x_i} \geq E_{x_i}$, nenhum tempo ocioso é inserido antes de x_i.

Se $C_{x_i} < E_{x_i}$, verifica-se se é vantajoso reduzir a antecipação da tarefa x_i. Para isso, seja $ult(x_i)$ a última tarefa do bloco que contém a tarefa x_i. Para reduzir a antecipação da tarefa x_i é necessário deslocar todas as tarefas do conjunto de tarefas $A = \{x_i, x_{i+1}, \ldots, ult(x_i)\}$ para a direita. O custo $CM(A)$ para deslocar as tarefas de A uma unidade de tempo para a direita é dado pela equação (3):

$$CM(A) = \sum_{j \in A:\ C_j \geq T_j} \beta_j - \sum_{j \in A:\ C_j < E_j} \alpha_j \tag{3}$$

Se $CM(A) \geq 0$, tem-se que não é vantajoso mover as tarefas do conjunto A para a direita, dado que o aumento em relação aos atrasos será maior que as reduções relativas às antecipações.

Se $CM(A) < 0$, tem-se que é vantajoso deslocar as tarefas do conjunto A para a direita. As tarefas de A são então deslocadas para a direita φ unidades de tempo, sendo φ dado pela equação (4).

$$\varphi = \begin{cases} \min(m_1, m_2)\ , \text{ se } ult(x_i) = x_n \\ \min\left(C_{ult(x_i)+1} - P_{ult(x_i)+1} - S_{(ult(x_i))(ult(x_i)+1)}, m_1, m_2\right)\ , \text{ se } ult(x_i) \neq x_n \end{cases} \tag{4}$$

em que $m_1 = \min_{j \in A:\ E_j \leq C_j < T_j} (T_j - C_j)$ e $m_2 = \min_{j \in A:\ C_j < E_j} (E_j - C_j)$

Após o deslocamento das tarefas de A φ unidades para a direita, os seguintes casos podem ocorrer:

Se $C_{ult(x_i)} + S_{(ult(x_i))(ult(x_i)+1)} + P_{ult(x_i)+1} = C_{ult(x_i)+1}$, tem-se a união de A ao bloco sucessor. Atualiza-se então $ult(x_i)$ e A e verifica-se se é vantajoso deslocar as tarefas do novo conjunto A para a direita (via análise de $CM(A)$).

Se $C_{ult(x_i)} + S_{(ult(x_i))(ult(x_i)+1)} + P_{ult(x_i)+1} < C_{ult(x_i)+1}$, verifica-se se ainda é vantajoso deslocar as tarefas de A para a direita (via análise de $CM(A)$).

O AAOTO é encerrado quando não for mais vantajoso inserir tempo ocioso antes das tarefas de X. O custo gerado por antecipação e atraso de uma tarefa i, concluída na data C_i, pode ser dado pela função $g_i(C_i) = \alpha_i \cdot \max\{0, E_i - C_i\} + \beta_i \cdot \max\{0, C_i - T_i\}$. A função $g_i(C_i + x)$ é uma função linear por partes e convexa em relação a x para todo $i \in \{1,\ 2,\ \cdots,\ n\}$ (Wan e Yen, 2002). Considere que há l blocos em X, ou seja, $X = B_1 \cup B_2 \cup \ldots \cup B_l$. Logo, a função custo de mover o bloco de tarefas B_j x unidades de tempo para a direita pode ser expressa pela equação (5):

$$Custo_j(x) = \sum_{i \in B_j} g_i(C_i + x), \tag{5}$$

em que $C_i + x$ é a nova data de conclusão do processamento da tarefa i.

Lema 1 ***(Wan e Yen, 2002):*** *O somatório de duas funções lineares por partes e convexas é também uma função linear por parte e convexa.*

Baseado no Lema 1, tem-se:

Proposição 1 ***(Wan e Yen, 2002):*** *$Custo_j(x)$ é uma função linear por partes e convexa em relação a x, para todo $j \in \{1,\ 2,\ \cdots,\ l\}$.*

Devido à natureza linear por partes e convexa da função custo, o custo mínimo do bloco B_j ocorre nos pontos extremos de sua função $Custo_j$, isto é, no início ou no final da janela de entrega de uma das tarefas no bloco. Tal fato, associado à Proposição 2, garante que o algoritmo apresentado determina os instantes ótimos de início do processamento das tarefas da sequência.

Proposição 2 ***(Wan e Yen, 2002):*** *O custo total de uma dada sequência de tarefas é ótimo se cada bloco B_j na sequência alcançar seu ponto mínimo, exceto s_{x_1}, que pode ser igual a zero.*

O Algoritmo 1 descreve o procedimento AAOTO aplicado a uma sequência $X = (x_1, x_2, \ldots, x_n)$ do conjunto I de n tarefas.

Para ilustrar a utilização do AAOTO, considere o problema de sequenciamento de 4 tarefas representado pela Tabela 1. Nesta tabela são apresentados o tempo de processamento (P_i), o custo por unidade de antecipação (α_i), o custo por unidade de atraso (β_i), a data de início (E_i) e de término (T_i) da janela de entrega, bem como os tempos de *setup* relativos a cada tarefa $i \in I$.

Tabela 1: Problema-teste para ilustração da aplicação do AAOTO.

	Dados					***setup***			
Tarefas	P	E	T	α	β	1	2	3	4
1	3	14	15	2	4	0	2	1	2
2	4	22	24	7	9	1	0	2	3
3	4	9	12	7	8	1	3	0	1
4	3	5	7	1	4	1	2	2	0

Dada a sequência $X = (3,\ 4,\ 1,\ 2)$, primeiramente todas as tarefas são programadas para iniciar o mais cedo possível, conforme a Figura 1. Nesse posicionamento, a soma das penalidades por antecipações e atrasos é igual a 71 unidades.

Na segunda etapa é verificado se é vantajoso inserir tempo ocioso antes de cada tarefa. Esse procedimento é feito iterativamente, começando pela última tarefa da sequência ($x_4 = 2$) e

Algoritmo 1 *AAOTO*(n, I, X)

$C_{x_1} \leftarrow P_{x_1}$;
para $i = 2, \ldots, n$ faça
  $C_{x_i} \leftarrow C_{x_{i-1}} + S_{(x_{i-1})(x_i)} + P_{x_i}$;
fim-para;
para $i = n, n-1, \ldots, 2, 1$ faça
  se $C_{x_i} < E_{x_i}$ então
    $ult(x_i) \leftarrow$ última tarefa do bloco que contém x_i;
    $A \leftarrow \{x_i, x_{i+1}, \ldots, ult(x_i)\}$;
    $CM(A) \leftarrow \sum_{j \in A\,:\; C_j \geq T_j} \beta_j - \sum_{j \in A\,:\; C_j < E_j} \alpha_j$;
    enquanto $CM(A) < 0$ faça
      $m_1 \leftarrow \min_{j \in A\,:\; E_j \leq C_j < T_j} (T_j - C_j)$; $m_2 \leftarrow \min_{j \in A\,:\; C_j < E_j} (E_j - C_j)$;
      $\varphi \leftarrow \begin{cases} \min(m_1, m_2), & \text{se } ult(x_i) = x_n \\ \min\left(C_{ult(x_i)+1} - P_{ult(x_i)+1} - S_{(ult(x_i))(ult(x_i)+1)}, m_1, m_2\right), & \text{se } ult(x_i) \neq x_n \end{cases}$;
      para $j \in A$ faça
        $C_j = C_j + \varphi$;
      fim-para;
      se $C_{ult(x_i)} + S_{(ult(x_i))(ult(x_i)+1)} + P_{ult(x_i)+1} = C_{ult(x_i)+1}$ então
        `// A é anexado ao bloco sucessor e atualizado.`
        $ult(x_i) \leftarrow$ última tarefa do bloco que contém x_i;
        $A \leftarrow \{x_i, x_{i+1}, \ldots, ult(x_i)\}$;
      fim-se
      $CM(A) \leftarrow \sum_{j \in A\,:\; C_j \geq T_j} \beta_j - \sum_{j \in A\,:\; C_j < E_j} \alpha_j$;
    fim-enquanto;
  fim-se
fim-para;
$f(X) \leftarrow \sum_{i=1}^{n} \alpha_{x_i} \cdot \max\{0, E_{x_i} - C_{x_i}\} + \beta_{x_i} \cdot \max\{0, C_{x_i} - T_{x_i}\}$;
Retorne $f(X)$;

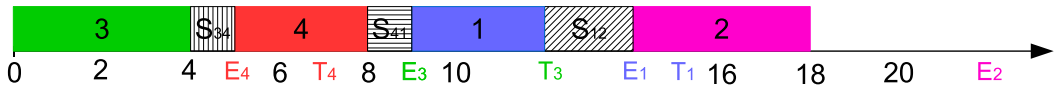

Figura 1: Posicionamento inicial do AAOTO.

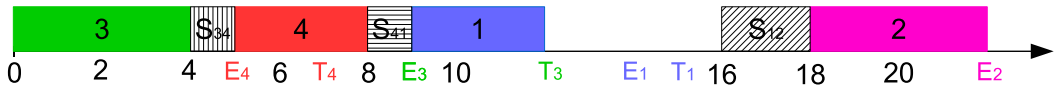

Figura 2: Posicionamento após a iteração 1 da segunda etapa do AAOTO.

terminando com a primeira ($x_1 = 3$). Como a inserção de tempo ocioso antes de $x_4 = 2$ reduz a sua antecipação ($CM(\{x_4\}) = -7 < 0$), x_4 é deslocada 4 unidades de tempo para a direita (Figura 2). Após esse deslocamento, a soma das penalidades por antecipações e atrasos é igual a 43.

Na iteração 2 da segunda etapa do AAOTO, verifica-se se é vantajoso inserir tempo ocioso antes da tarefa $x_3 = 1$. Como tal inserção de tempo é vantajosa, pois $CM(\{x_3\}) = -2 < 0$, então x_3 é deslocada 2 unidades de tempo para a direita (Figura 3). Após esse deslocamento, a soma das penalidades por antecipações e atrasos é igual a 39 unidades.

Na iteração 3 da segunda etapa do AAOTO, verifica-se se é vantajoso inserir tempo ocioso

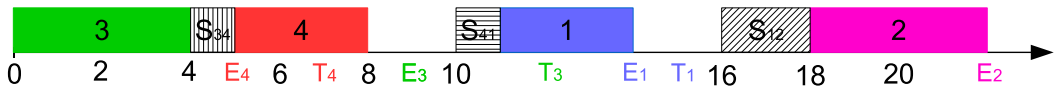

Figura 3: Posicionamento após a iteração 2 da segunda etapa do AAOTO.

antes da tarefa $x_2 = 4$. Como tal inserção de tempo ocioso não é vantajosa ($CM(\{x_2\}) = 4 \geq 0$), passa-se para a iteração 4. Na iteração 4 analisa-se se é vantajoso inserir tempo ocioso antes da primeira tarefa da sequência ($x_1 = 3$). Para que essa inserção de tempo seja possível, é necessário deslocar o bloco de tarefas $(x_1, x_2) = (4, 3)$ para a direita. Como esse deslocamento é vantajoso ($CM(\{x_1, x_2\}) = -7 + 4 = -3 < 0$), o bloco (x_1, x_2) é deslocado 2 unidades de tempo para a direita. Após esse deslocamento, a soma das penalidades por antecipações e atrasos é igual a 33. Forma-se, então, um novo bloco $(x_1, x_2, x_3) = (3, 4, 1)$ (Figura 4). Nesse momento, é necessário verificar se esse novo bloco formado está na posição ótima. Como o deslocamento desse bloco para a direita é vantajoso ($CM(\{x_1, x_2, x_3\}) = -7 + 4 + 0 = -3 < 0$), ele é deslocado 1 unidade de tempo para a direita (Figura 5). Após esse deslocamento, a soma das penalidades por antecipações e atrasos é igual a 30. Já que a inserção de mais tempos ociosos antes da primeira tarefa não é vantajosa ($CM(\{x_1, x_2, x_3\}) = -7 + 4 + 4 = 1 \geq 0$), tem-se que esse posicionamento é ótimo para a sequência X.

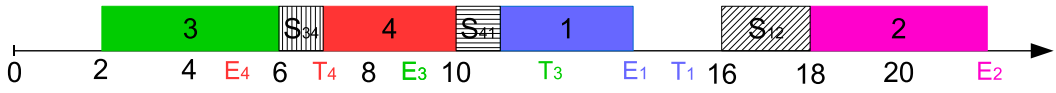

Figura 4: Posicionamento após a iteração 4 da segunda etapa do AAOTO.

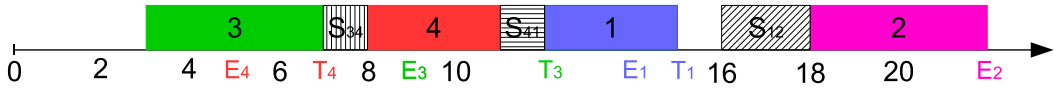

Figura 5: Posicionamento ótimo para a sequência dada.

4. Enumeração implícita

Nesta seção é proposto um algoritmo de enumeração implícita para resolver o PSUMAA, doravante denotado por EI. Para auxiliar sua apresentação, seja I o conjunto de n tarefas que devem ser sequenciadas. Seja UB um limite superior dado para o problema, obtido de forma heurística, por exemplo. Durante o processo de enumeração, cada nó δ da árvore armazena uma subsequência de i tarefas ($0 \leq i \leq n$), bem como o conjunto das $n - i$ tarefas que estão fora de tal subsequência, representados por $seq(\delta)$ e $nseq(\delta)$, respectivamente.

Seja L a lista de nós a serem investigados. Uma sequência de tarefas X é avaliada pela função f dada pela Eq. (1).

O algoritmo EI é inicializado com a inserção do nó δ_0 tal que $seq(\delta_0) = \varnothing$ e $nseq(\delta_0) = I$ como o único nó de L. A partir desse momento, cada iteração de EI consiste nos seguintes passos:

- Seja δ o último nó que foi inserido em L;
- Retira-se de L o nó δ;
- Se $nseq(\delta) \neq \varnothing$, para cada tarefa $j \in nseq(\delta)$, verifica-se se a subsequência $seq(\delta) \cup (j)$, obtida pela inserção da tarefa j no final da subsequência $seq(\delta)$, viola a condição de otimalidade (apresentada a seguir). Caso tal condição não seja violada, o nó δ_{filho} que armazena a subsequência $seq(\delta) \cup (j)$ e o conjunto $nseq(\delta) \backslash \{j\}$ é inserido em L.
- Se $nseq(\delta) = \varnothing$, tem-se que a subsequência $seq(\delta)$ corresponde à uma solução completa. Se $f(seq(\delta)) < UB$, atualiza-se o valor de UB para $UB = f(seq(\delta))$.

O algoritmo EI é interrompido quando $L = \varnothing$ e o valor final de UB for a solução ótima do problema. Para verificar se uma dada subsequência de tarefas $X = \{x_1, x_2, \cdots, x_u\}$, com $1 \leq u \leq n$, pode estar contida em uma solução ótima, ou seja, se ela não viola a condição de otimalidade, primeiramente calcula-se $f(X)$ e determina-se quais das tarefas de X estão antecipadas e quais estão atrasadas. Isso é feito aplicando-se o algoritmo AAOTO (ver Seção 3). A subsequência X viola a condição de otimalidade se um dos seguintes casos ocorrerem:

i) $f(X) > UB$, dado que a inserção de novas tarefas no final de X nunca irá diminuir $f(X)$;

ii) Se existem duas tarefas adjacentes em X, x_k e x_{k+1}, tais que $C_{x_{k+1}} \leq \min(E_{x_k}, E_{x_{k+1}})$ e $P_{x_k}\alpha_{x_{k+1}} < P_{x_{k+1}}\alpha_{x_k}$, então trocar a ordem de execução das tarefas x_k e x_{k+1} reduz as penalidade associadas à sequência X. A inserção de tarefas no final X não fará que x_k e x_{k+1} sejam deslocadas de modo que esse cenário seja alterado, ou seja, a subsequência obtida pela troca mencionada sempre poderá dar origem à sequências de melhor qualidade.

iii) Se a primeira tarefa inicia sua execução na data 0 ($C_{x_1} = P_{x_1}$) e existem duas tarefas adjacentes em X, x_k e x_{k+1}, tais que não há nenhum tempo ocioso inserido antes delas e $custo1 > custo2$, sendo $custo1$ e $custo2$ dados por:

$$custo1 = \alpha_{x_k} e_{x_k} + \beta_{x_k} t_{x_k} + \alpha_{x_{k+1}} e_{x_{k+1}} + \beta_{x_{k+1}} t_{x_{k+1}} \text{ e}$$

$$custo2 = \alpha_{x_k} \max(0, E_{x_k} - C_{x_k} - P_{x_{k+1}}) + \beta_{x_k} \max(0, C_{x_k} + P_{x_{k+1}} - T_{x_k}) + \\ + \alpha_{x_{k+1}} \max(0, E_{x_{k+1}} - C_{x_{k+1}} + P_{x_k}) + \beta_{x_{k+1}} \max(0, C_{x_{k+1}} - P_{x_k} - T_{x_{k+1}}).$$

Observa-se que, neste caso, trocar a ordem de execução das tarefas x_k e x_{k+1} reduz as penalidade associadas à sequência X. Como a inserção de novas tarefas no final de X não mudaram esse cenário, dado que não é possível deslocar x_k e x_{k+1} para a esquerda, tem-se que a subsequência obtida pela troca mencionada sempre poderá dar origem a sequências de melhor qualidade.

5. GVNS aplicado ao problema

Nesta seção é apresentado o algoritmo heurístico *General Variable Neighborhood Search* (GVNS) para resolver o PSUMAA. Em seu pseudocódigo, apresentado no Algoritmo 2, $GVNSmax$ é o número máximo de iterações sem melhora na melhor solução encontrada e define o critério de parada do algoritmo. O detalhamento do GVNS proposto neste trabalho é feito nas subseções seguintes.

Algoritmo 2 GVNS$(I, n, f(.), N^1(.), N^2(.), N^3(.), GVNSmax)$

```
X ← SolucaoInicial(I, n);
f* ← f(X);  Iter ← 0;
enquanto Iter < GVNSmax faça
    Iter ← Iter + 1;  k ← 1;
    enquanto k ≤ 3 faça
        Gere, aleatoriamente, um vizinho X' ∈ N^k(X);
        X'' ← VND(X', f(.), N^1(.), N^2(.), N^3(.), VNDmax);
        se f(X'') < f(X) então
            X ← X'';  f* ← f(X);  k ← 1;  Iter ← 0;
        senão
            k ← k + 1;
        fim-se
    fim-enquanto;
fim-enquanto;
Retorne f*;
```

5.1. Representação de uma solução

Uma solução (sequência) para o PSUMAA de n tarefas é representada por um vetor X de n posições, com cada posição $i = 1, 2, \cdots, n$ indicando a ordem de execução da tarefa x_i. Por ex., na sequência $X = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$, a tarefa 5 é a primeira a ser executada e a 6 é a última.

5.2. Solução inicial

Uma solução inicial para o PSUMAA é construída por meio da aplicação da heurística construtiva gulosa EDD (*Earliest Due Date*), frequentemente utilizada na literatura para o caso de datas de entregas distintas. A construção proposta inicia com todas as tarefas fora da sequência e, iterativamente, a tarefa que possui a janela de entrega com menor data de início e ainda não sequenciada é inserida no final da subsequência corrente. O procedimento de construção é interrompido quando não houver mais tarefa a ser sequenciada.

5.3. Vizinhança e avaliação de uma solução

Para explorar o espaço de soluções são usados três tipos de movimentos: troca da ordem de processamento de duas tarefas da sequência de execução, realocação de uma tarefa para outra posição da sequência e realocação de um bloco de k tarefas para outra posição da sequência. Esses movimentos definem, respectivamente, as vizinhanças N^1, N^2 e N^3.

Como os movimentos utilizados não geram soluções inviáveis, uma sequência X é avaliada pela função f dada pela Eq. (1), a qual deve ser minimizada. Os valores de e_i e t_i correspondentes a cada tarefa $i \in I$ da sequência X são determinados pelo algoritmo AAOTO da Seção 3.

5.4. VND

A busca local utilizada no GVNS é feita pelo procedimento *Variable Neighborhood Descent* – VND (Mladenović e Hansen, 1997). Neste trabalho, o VND utiliza a seguinte ordem de buscas locais: 1) BL1: Descida randômica usando vizinhança N^1; 2) BL2: Descida randômica usando vizinhança N^2 e 3) BL3: Descida randômica usando vizinhança N^3.

Na primeira busca local (BL1), duas tarefas são escolhidas aleatoriamente e a ordem de seus processamentos na sequência de execução é trocada. Se a nova sequência produzir uma solução de melhora, ela é aceita e passa a ser a solução corrente; caso contrário, é testada outra troca aleatória. A BL1 termina quando ocorrer *VNDmax* trocas consecutivas sem melhora na solução corrente, sendo *VNDMax* um parâmetro do procedimento.

Na busca BL2, aleatoriamente escolhe-se uma tarefa na sequência e uma nova posição para ela. Se a nova sequência produzir uma solução melhor, a nova sequência passa a ser a solução corrente e volta-se à busca local BL1; caso contrário, é testada outra realocação aleatória. A BL2 é interrompida após *VNDMax* realocações consecutivas sem melhora na solução corrente.

Na busca local BL3, aleatoriamente escolhe-se um bloco de tarefas na sequência de execução (o tamanho do bloco também é escolhido de forma aleatória no intervalo $[1,\ n\ -\ 1\ ]$) e uma nova posição para ele. Se a nova sequência produzir uma solução de melhora, ela passa a ser a solução corrente e volta-se à busca local BL1; caso contrário, é testada outra realocação de bloco aleatória. A BL3 é interrompida após *VNDMax* realocações de blocos consecutivas sem melhora na solução corrente. Neste último caso, o VND é encerrado e a melhor solução encontrada é retornada.

6. Resultados computacionais

Os algoritmos AAOTO, GVNS e EI apresentados nas seções 3, 5 e 4, respectivamente, foram implementados na linguagem C++, usando o compilador NetBeans IDE 7.4. Para testar tais algoritmos, foi utilizado o conjunto de problemas-teste de Rosa e Souza (2009) para o PSUMAA com tempos de *setup* dependentes da sequência de execução. O conjunto utilizado encontra-se disponível em `http://www.decom.ufop.br/prof/marcone/projects/scheduling/instances.htm` e é composto por problemas-teste de 6 a 20 tarefas, sendo 16 problemas-teste em cada grupo de problemas de mesmo tamanho. Como o algoritmo EI não considera tempos de *setup*, as tabelas de tempos de *setup* de cada problema-teste foram desconsideradas.

Os testes foram realizados em um computador Intel® Core™ i7-3632QM CPU, 2.20 GHz, com 8 GB de RAM e sistema operacional ubuntu 13.10. Apesar de o processador desse equipamento possuir mais de um núcleo, os algoritmos não foram otimizados para multiprocessamento.

O desempenho do AAOTO foi comparado com o do algoritmo ADDOIP de Gomes Júnior et al. (2007), o qual também foi implementado nas mesmas condições. Os resultados obtidos são apresentados na Tabela 2. Nessa tabela, a primeira coluna indica o número de tarefas dos problemas-teste de cada grupo. A coluna AAOTO apresenta os tempos médios, em segundos, demandados ao utilizar o AAOTO para avaliar cada sequência de tarefas gerada pelos algoritmos GVNS e EI. Já a coluna ADDOIP apresenta os tempos médios demandados (em segundos) ao utilizar o algoritmo ADDOIP para avaliar as sequências de tarefas geradas nos algoritmos GVNS e EI.

Para validar os procedimentos apresentados, a formulação de programação matemática de Gomes Júnior et al. (2007) para representar o PSUMAA foi implementada com o auxílio da Concert Technology C++ do otimizador IBM ILOG CPLEX Optimization Studio 12.5.1. Como não foi utilizado nenhum ajuste para processamento paralelo nas implementações dos algoritmos GVNS e EI, o CPLEX foi configurado para utilizar apenas uma *thread* (as demais configurações não foram alteradas). Foram resolvidos os problemas-teste com até 11 tarefas e os tempos médios demandados para cada grupo de problemas de mesmo tamanho são apresentados na coluna CPLEX da Tabela 2. O tempo demandado por esse otimizador para resolver os problemas com mais de 11 tarefas se mostrou proibitivo.

Os parâmetros do algoritmo GVNS foram fixados empiricamente nos valores $GVNSmax = 2n$ e $VNDmax = 4n$, sendo n o número de tarefas a serem sequenciadas. Cada problema-teste foi resolvido 30 vezes tanto pelo algoritmo GVNS utilizando o ADDOIP para avaliar as sequências geradas, quanto pelo algoritmo GVNS utilizando o AAOTO para avaliar tais sequências. Os tempos médios, em segundos, demandados nas execuções de problemas-teste de mesmo tamanho são apresentados na Tabela 2. Observa-se que, conforme esperado, o GVNS encontrou sempre a mesma solução em todas as execuções de cada problema-teste. Observa-se, também, que o GVNS encontrou as soluções ótimas dos problemas-teste em que tais soluções são conhecidas.

Para cada problema-teste utilizou-se a solução obtida pelo GVNS para calcular o limite superior inicial (UB) do algoritmo EI. Desse modo, o algoritmo EI foi capaz de resolver todos os problemas-teste com até 15 tarefas em tempo computacional viável. O tempo demandado pelo algoritmo EI para resolver os problemas-teste com mais de 15 tarefas se mostrou proibitivo. As médias dos tempos demandados (em segundos) pelo algoritmo EI com o algoritmo ADDOIP e do algoritmo EI com o AAOTO são apresentadas na Tabela 2.

Tabela 2: Comparação ADDOIP × AAOTO (tempos médios em segundos).

Nº de	**ADDOIP**		**AAOTO**		**CPLEX**
Tarefas	**GVNS**	**EI**	**GVNS**	**EI**	
06	0,01	0,00	0,00	0,00	0,04
07	0,03	0,00	0,00	0,00	0,38
08	0,04	0,02	0,01	0,01	2,62
09	0,07	0,09	0,01	0,04	18,91
10	0,11	0,44	0,02	0,18	183,67
11	0,17	2,61	0,03	0,97	2085,10
12	0,27	19,45	0,04	6,12	–
13	0,39	212,35	0,05	66,76	–
14	0,54	1126,28	0,07	277,56	–
15	0,72	6594,25	0,10	1799,71	–
16	1,06	–	0,11	–	–
17	1,43	–	0,15	–	–
18	1,79	–	0,20	–	–
19	2,37	–	0,22	–	–
20	3,31	–	0,29	–	–

De acordo com a Tabela 2, os tempos médios demandados pelo algoritmo GVNS com o AAOTO foram sempre menores que os respectivos tempos médios demandados pelo algoritmo GVNS com o algoritmo ADDOIP. Para o grupo de problemas-teste com 20 tarefas, por exemplo, o tempo médio demandado pelo GVNS/ADDOIP foi mais de 11 vezes maior que o tempo médio demandado pelo GVNS/AAOTO. Para os problemas-teste com até 11 tarefas, o GVNS foi capaz de encontrar as respectivas soluções ótimas obtidas pelo CPLEX em tempos computacionais consideravelmente inferiores à tal otimizador, independente do algoritmo utilizado para avaliar as sequências geradas. Para o grupo de problemas-teste com 11 tarefas, por exemplo, enquanto o GVNS/AAOTO demandou tempo médio de 0,03 segundos, o CPLEX demandou 2085,1 segundos.

Na Tabela 2 também é possível observar que o algoritmo EI resolveu de forma ótima todos os problemas-teste com até 15 tarefas, ao passo que o CPLEX resolveu somente os com até 11 tarefas. Além disso, os tempos médios demandados pelo CPLEX foram bem superiores àqueles demandados pelo EI. Para o grupo de problemas-teste com 11 tarefas, por exemplo, enquanto o EI/AAOTO demandou tempo médio menor que 1 segundo, o CPLEX demandou 2085,1 segundos. Os tempos médios demandados pelo algoritmo EI/AAOTO foram sempre menores que aqueles requeridos pelo algoritmo EI/ADDOIP. Para o grupo de problemas-teste com 14 tarefas, por exemplo, o tempo médio demandado pelo EI/ADDOIP foi mais de 4 vezes maior que o tempo médio demandado pelo EI/AAOTO. Apesar de não ser apresentado na Tabela 2, observa-se que se o tempo de solução fosse limitado a 1 hora, o algoritmo EI/ADDOIP não seria capaz de resolver um dos problemas-teste com 14 tarefas e 6 dos problemas-teste com 15 tarefas. Nessa mesma condição, o algoritmo EI/AAOTO não resolveria somente 3 dos problemas-teste com 15 tarefas.

7. Conclusões

Este trabalho tratou o problema de sequenciamento de tarefas em uma máquina com penalidades por antecipação e atraso da produção (PSUMAA), considerando a existência de janelas de entrega distintas.

Propôs-se um novo algoritmo de alocação ótima de tempos ociosos (AAOTO) em uma dada sequência de execução. Além disso, foram propostos, também, um algoritmo de enumeração implícita (EI) e um algoritmo GVNS (*General Variable Neighborhood Search*) para resolver o problema. Cada sequência gerada pelos algoritmos EI e GVNS é avaliada com auxílio do AAOTO.

Os algoritmos EI e GVNS associados ao AAOTO foram utilizados para resolver um conjunto de problemas-teste da literatura. Experimentou-se também a utilização do algoritmo de determinação de datas ótimas de início de processamento – ADDOIP (Gomes Júnior et al., 2007) para avaliar as sequências geradas pelos algoritmos EI e GVNS. O AAOTO se mostrou mais eficiente que o ADDOIP, visto que os tempos médios demandados pelos algoritmos EI e GVNS associados ao AAOTO foram significativamente inferiores aos respectivos tempos demandados por esses algoritmos associados ao ADDOIP.

O algoritmo EI se mostrou uma boa opção para resolução exata do PSUMAA, pois foi capaz de resolver o PSUMAA em tempo bem menor que aquele requerido pelo CPLEX aplicado à formulação de programação inteira mista de Gomes Júnior et al. (2007). Por outro lado, o algoritmo GVNS mostrou-se robusto, encontrando a solução ótima de todos os problemas-teste para os quais tal solução é conhecida e variabilidade nula nos demais casos.

Agradecimentos

Os autores agradecem à CAPES, à FAPEMIG e ao CNPq, bem como ao CEFET-MG e à UFOP, pelo apoio ao desenvolvimento deste trabalho.

Referências

Allahverdi, A.; Gupta, J. N. D. e Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, v. 27, p. 219–239.

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

França Filho, M. F. *GRASP e Busca Tabu aplicados a problemas de programação de tarefas em máquinas paralelas*. Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas, (2007).

Glover, F. e Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Boston.

Gomes Júnior, A. C.; Carvalho, C. R. V.; Munhoz, P. L. A. e Souza, M. J. F. (2007). Um método heurístico híbrido para a resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. *Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional*, p. 1649–1660, Fortaleza (CE).

Gonçalves, F. A. C. A. e Souza, M. J. F. (2010). Sequenciamento em uma máquina: otimização heurística via multiprocessamento paralelo. *Anais do XLII Simpósio Brasileiro de Pesquisa Operacional*, p. 1687–1698, Bento Gonçalves (RS).

Koulamas, C. (1996). Single-machine scheduling with time windows and earliness/tardiness penalties. *European Journal of Operational Research*, v. 91, p. 190–202.

Lee, C. Y. e Choi, J. Y. (1995). A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. *Comp. & Ops. Research*, v. 22, n. 8, p. 857–869.

Liaw, C.-F. (1999). A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers & Operations Research*, v. 26, p. 679–693.

Lourenço, H. R.; Martin, O. C. e Stützle, T. (2003). Iterated local search. Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, p. 321–353. Kluwer Academic Publishers.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, v. 24, n. 11, p. 1097–1100.

Penna, P. H. V.; Souza, M. J. F.; Gonçalves, F. A. C. A. e Ochi, L. S. (2012). Uma heurística híbrida para minimizar custos com antecipação e atraso do sequenciamento da produção em uma máquina. *Produção*, v. 22, p. 766–777.

Ribeiro, F. F.; Souza, M. J. F. e Souza, S. R. (2010). An adaptive genetic algorithm to the single machine scheduling problem with earliness and tardiness penalties. Rocha Costa, A. C.; Vicari, R. M. e Tonidandel, F., editors, *Advances in Artificial Intelligence - SBIA 2010*, volume 6404 of *Lecture Notes in Computer Science*, p. 203–212.

Rosa, B. F. e Souza, M. J. F. (2009). Uma nova formulação de programação matemática indexada no tempo para uma classe de problemas de seqüenciamento em uma máquina. *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional*, p. 2898–2909, Porto Seguro (BA).

Rosa, B. F.; Souza, M. J. F. e Souza, S. R. (2010). Uma heurística de redução do espaço de busca para uma classe de problemas de sequenciamento de tarefas em uma máquina. *Anais do XVIII Congresso Brasileiro de Automática*, p. 3583–3590, Bonito (MS).

Souza, M. J. F.; Ochi, L. S. e Maculan Filho, N. (2008). Minimizing earliness and tardiness penalties on a single machine scheduling problem with distinct due windows and sequence-dependent setup times. *Proceedings of the ALIO/EURO 2008 Conference*, volume 1, p. 1–6, Buenos Aires.

Wan, G. e Yen, B. P.-C. (2002). Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *EJOR*, v. 142, p. 271–281.