ALGORITMO DE PROGRAMAÇÃO DE MÁQUINAS INDIVIDUAIS COM PENALIDADES DISTINTAS DE ADIANTAMENTO E ATRASO

Emerson C. Colin
Tamio Shimizu
Escola Politécnica da USP
Departamento de Engenharia de Produção

Resumo

Neste trabalho consideramos o problema de máquina única, com datas de entrega e penalidades de adiantamento e atraso distintas para cada ordem. Considerando que a seqüência seja predefinida, o objetivo a ser alcançado é a minimização da soma das diferenças (adiantamentos ou atrasos) penalizadas das ordens. Este trabalho é apresentado como uma generalização do algoritmo de programação de Garey *et al*. (1988). Através de uma estrutura computacional denominada fila de prioridade, este novo algoritmo permite a elaboração de um programa em tempo $O(n\log n)$, enquanto que o melhor encontrado na literatura atualmente é de tempo $O(n^2)$.

Palavras-chave: programação da produção, inserção de ociosidade, programação JIT.

Abstract

In this work we consider the one machine problem, with distinct due-dates and penalties for earliness and tardiness. For a previously defined sequence, we utilize the sum of weighted lateness (earliness or tardiness) as objective function. This work is presented as a generalization of the Garey *et al.* (1988) scheduling algorithm. Using a computational structure called heap, this algorithm allows a schedule construction in $O(n\log n)$ time while the best found in literature runs in $O(n^2)$.

Keywords: production scheduling, idle time insertion, JIT scheduling.

1. INTRODUÇÃO

Desde o início da difusão de princípios do JIT (*Just-In-Time*) – que pode ser considerado como um sistema de administração industrial relativo ao estoque –, tem crescido a importância da diminuição do estoque no processamento de produtos. Estamos fazendo mais um esforço nesse sentido, considerando como característica-chave de nosso trabalho um dos elementos mais importantes do JIT – evitar que produtos e componentes sejam produzidos ou entregues antes da data correta (Groenevelt (1993)). Embora elementos como os erros, a diminuição do tempo de preparação, a limpeza e a organização da fábrica também façam parte do JIT, quer seja diretamente ou indiretamente, não estaremos considerando-os.

Sob o enfoque de uma das linhas de pesquisa da programação da produção (Baker & Scudder (1990)), o JIT pode ter sua dimensão de programação caracterizada através da penalização do término adiantado de ordens de produção. Devemos lembrar que embora a programação do JIT geralmente esteja associada ao *kanban*, muitos estudos teóricos têm sido feitos no sentido de utilizar a penalização do término adiantado de ordens em sistemas que empurram a produção. Para boas discussões acerca da aplicabilidade e definição de ambos os sistemas, vide Silver *et al*. (1998) e Vollmann *et al*. (1997).

Uma das conseqüências mais importantes advindas das filosofias JIT é que dependendo de certas condições, pode valer a pena manter uma máquina parada, inserindo-se tempo de ociosidade entre a realização de duas ordens. Essa característica, peculiar ao ambiente do tipo JIT, é válida enquanto os custos provenientes de um suposto atraso não forem maiores do que os custos provenientes do adiantamento. Portanto, uma metodologia que leve em consideração ambos os custos – de adiantamento e de atraso – pode ser de grande validade quando se deseja trabalhar num ambiente desse tipo. Para um maior esclarecimento com relação a esse assunto, ver Colin (1997), onde o algoritmo apresentado neste trabalho foi utilizado inicialmente.

Na próxima seção fazemos uma breve revisão sobre as principais definições utilizadas neste trabalho e definimos formalmente nosso problema. Na seção 3 fazemos algumas discussões acerca de outros trabalhos que trataram do mesmo tipo de problema. Na seção 4 apresentamos o algoritmo, enquanto que na seção 5 discutimos características de sua implementação computacional. Finalmente na seção 6 apresentamos um exemplo numérico do uso do algoritmo.

2. CONCEITOS PRELIMINARES E DEFINIÇÃO DO PROBLEMA

Para o desenvolvimento do trabalho, utilizamos definições e notações clássicas, de acordo com Morton & Pentico (1993). A seqüenciação é considerada como um ordenamento das ordens de produção. Para um conjunto de ordens $J=\{J_1,...,J_n\}$, deve-se encontrar a seqüência $s=<s(1),s(2),...,s(n)>$ com $s(j)\in\{1,2,...,n\}$ e $s(j)\neq s(i)$ sempre que $j\neq i$. Cada ordem J_j (j=1,...,n), possui quatro valores inteiros associados $\{p_j,d_j,w_j,h_j\}$ que representam o tempo de processamento, a data de entrega, a penalidade de atraso e a penalidade de adiantamento respectivamente. A programação é definida como a determinação dos horários de início e de término das ordens de uma dada seqüência. Para uma dada ordem J_j, existe um intervalo de processamento $[e_j,e_j+p_j]$ onde e_j e p_j significam respectivamente o horário efetivo de início do processamento e o tempo de processamento da ordem J_j. No caso do problema de uma máquina, para duas ordens consecutivas $J_{s(j)}$ e $J_{s(j+1)}$, com intervalos de processamento definidos por $[e_{s(j)},e_{s(j)}+p_{s(j)}]$ e $[e_{s(j+1)},e_{s(j+1)}+p_{s(j+1)}]$, poderá haver intersecção entre os dois intervalos apenas nos pontos extremos dos intervalos. Seja $W_{s(j)}$ o período de ociosidade antes da realização da ordem $J_{s(j)}$. O programa $p(s)$ de um conjunto de ordens J, é a determinação do

horário efetivo de início de cada ordem, ou seja, $p(s)=<e_{s(1)},e_{s(2)},...,e_{s(n)}>$. Um programa parcial, $p_j(s)$, é um programa que contém as j primeiras ordens da seqüência s.

Estamos considerando que a seqüência já foi definida preliminarmente, e portanto podemos omitir a seqüência s de nossa notação. A função diferença é definida como $L_j=C_j-d_j$, onde $C_j=e_j+p_j$ é o horário de término da ordem J_j. Costuma-se separar a função diferença quando ela é positiva e quando ela é negativa. Se negativa, é chamada de adiantamento, e é definida como $E_j=\max(0,-L_j)$, enquanto que se positiva, é chamada de atraso, e pode ser definida como $T_j=\max(0,L_j)$.

Estamos considerando o caso de uma única máquina que deve processar uma seqüência de ordens $s=<J_1,J_2,...,J_n>$. O programa é realizado de maneira a tentar cumprir as datas de entrega d_j, sabendo-se que cada ordem demanda um tempo p_j para o seu processamento. Há possibilidade de inserção de tempo ocioso W_j antes da elaboração de uma certa ordem J_j. Para penalidades $h_j>0$ e $w_j>0$, a função-objetivo em questão pode ser definida como $\min g(p) = \sum_{j=1}^{n}(h_j E_j + w_j T_j)$.

3. INSERÇÃO DE OCIOSIDADE

O primeiro trabalho a apresentar um algoritmo para inserção de ociosidade pertence a Fry *et al.* (1987). De uma maneira bastante clara, os autores formularam o problema de inserção de ociosidade no problema de atraso e adiantamento com penalidades individuais, $\Sigma(h_jE_j+w_jT_j)$, como um problema de programação linear. Pela característica do tipo de problema, a solução acontece em tempo $O(n^2)$. Algum tempo depois, Davis & Kanet (1993) propuseram um outro algoritmo com tempo de solução no pior caso $O(n^2)$ para a solução do mesmo problema.

De uma maneira provavelmente independente, Garey *et al.* (1988) também propuseram um algoritmo de inserção de ociosidade no problema de atraso e adiantamento sem penalidades, $\Sigma(E_j+T_j)$, com tempo $O(n\log n)$. Para o caso da utilização de uma penalidade única para adiantamento e atraso, $\Sigma w_j(E_j+T_j)$, os autores indicam como o algoritmo deveria ser formulado. Yano & Kim (1991) elaboraram um algoritmo de inserção de ociosidade baseado na programação dinâmica. A inserção de ociosidade é feita no problema com função-objetivo $\Sigma(h_jE_j+w_jT_j)$, onde é suposto que $w_j\geq h_j\geq 0$. O tempo de solução no pior caso é $O(n^2\log n)$.

Todos os trabalhos supõem que os algoritmos sejam utilizados em uma seqüência que foi definida preliminarmente.

4. ALGORITMO DE INSERÇÃO DE OCIOSIDADE

Nesta seção fazemos uma adaptação no algoritmo de inserção de ociosidade de Garey *et al.* (1988) para a função-objetivo do tipo $\Sigma(h_jE_j+w_jT_j)$. Esta seção e a próxima seguem de forma bastante semelhante o desenvolvimento e a notação utilizada por Garey *et al*.

O horário desejado de início da ordem J_j é definido como $a_j=d_j-p_j$. O horário efetivo de início de uma dada ordem é denotado por e_j. Um bloco de ordens B_i (i=1,2,...,l) é definido como uma seqüência parcial que é programada sem inserção de ociosidade entre as ordens da mesma. Matematicamente pode-se dizer que uma dada seqüência de ordens $M =< J_{m_1}, J_{m_2},...,J_{m_j} >$ é

um bloco de ordens se $e_k+p_k=e_{k+1}$ (para $m_1 \le k < m_j$), $e_{m_1-1} + p_{m_1-1} < e_{m_1}$ (para $m_1>1$) e $e_{m_j} + p_{m_j} < e_{m_{j+1}}$ (para $m_j<n$).

Assumimos que um programa completo $p_n(s)$ possui l blocos $<B_1,...,B_l>$ ($1 \le i \le l$; $l \le n$). Um esquema gráfico dos conceitos de bloco é apresentado na Figura 1. Cada bloco B_i é particionado em 2 subconjuntos, denominados *Atrasado*(i) e *Adiantado*(i). Seja uma ordem $J_j \in B_i$. $J_j \in Atrasado(i)$ se $e_j > a_j$ e $J_j \in Adiantado(i)$ se $e_j \le a_j$, ou seja, se $J_j \in Atrasado(i)$, reduzindo-se e_j, reduz-se o atraso de J_j e se $J_j \in Adiantado(i)$, reduzindo-se e_j, aumenta-se o adiantamento de J_j. Adicionalmente define-se $H(i) = \sum_{J_j \in Adiantado(i)} h_j$ e $W(i) = \sum_{J_j \in Atrasado(i)} w_j$.

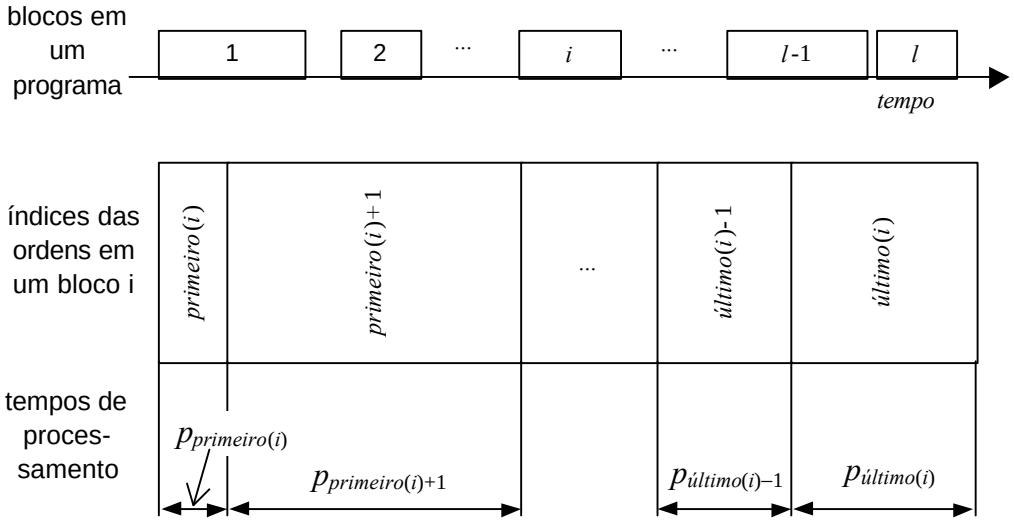

Figura 1: Significado gráfico dos blocos de um programa

Para se representar um bloco B_i, serão necessários apenas as definições dos índices extremos, ou seja, *primeiro*(i) que é o índice do primeiro elemento de B_i e *último*(i) que é o índice do último elemento de B_i. As ordens intermediárias são facilmente identificadas pois são relacionadas com a primeira e com a última.

4.1. VISÃO GERAL DO ALGORITMO

A idéia central do algoritmo está na construção e movimentação dos blocos, onde essa movimentação acontece sempre fazendo com que as ordens do mesmo sejam antecipadas. Sempre acontece uma antecipação do bloco pois as ordens adicionadas ao bloco estão sempre atrasadas ou no horário desejado. Se as datas de entrega forem suficientemente dispersas, cada bloco será formado por apenas uma ordem. Como isso obviamente não ocorre sempre, existem períodos de conflito, onde a capacidade disponível na máquina não é suficiente para processar todas as ordens de modo que suas datas de entrega sejam atendidas. Quando isso acontece, os adiantamentos e atrasos penalizados das ordens do bloco em consideração são avaliados procurando-se por uma oportunidade de se diminuir a penalidade total. Deve-se observar que um bloco contém ordens atrasadas e adiantadas e uma movimentação do bloco para mais cedo faz com que as ordens adiantadas fiquem mais adiantadas, enquanto que as ordens atrasadas atrasem menos ou passem a estar adiantadas. Neste trabalho uma ordem que se inicia no horário desejado é considerada adiantada.

Para esse fim, o algoritmo é iniciado fazendo com que a primeira ordem seja finalizada em sua data de entrega. Isso pode não ser possível, visto que a ordem já pode estar virtualmente atrasada − mesmo que ela seja iniciada no instante inicial, ela vai atrasar. Verifica-se se a próxima ordem pode ser terminada em sua data de entrega. Aqui pode acontecer duas coisas: um novo bloco ser formado ou a ordem ser adicionada ao último bloco formado. Um novo bloco será formado caso o término da última ordem do último bloco seja anterior ao início desejado da ordem em consideração. Por outro lado, a ordem será incluída no último bloco formado, caso o término da última ordem do último bloco seja posterior ou igual ao horário de início desejado da ordem em consideração. Neste último caso, haverá uma ponderação entre todas as ordens pertencentes ao bloco formado com a nova ordem. Se a nova ordem adicionada estiver atrasada e esse atraso, juntamente com os outros atrasos penalizados pertencentes ao bloco for maior que os adiantamentos penalizados, então o bloco é movimentado para mais cedo enquanto a penalização total diminui. Este procedimento é realizado até a última ordem ser programada.

4.2. DETALHAMENTO DO ALGORITMO

Para um melhor entendimento do algoritmo, ele pode ser considerado como tendo duas partes: a primeira, para construção de blocos e a segunda, para antecipação de blocos. De maneira mais precisa, podemos dizer que a primeira parte do algoritmo − construção de blocos − tem início simplesmente programando-se o horário efetivo de início de J_1 igual ao horário desejado de início de J_1, isto é, $e_1=a_1$. Para um programa parcial $p_j(s)$ já realizado, programa-se J_{j+1} conforme dois casos:

Caso 1: $e_j+p_j \leq a_{j+1}$. Programa-se J_{j+1} para começar em a_{j+1}. Nesse caso J_{j+1} não possui nem atraso nem adiantamento pois é inserido um período de ociosidade e portanto $g(p_{j+1}(s))=g(p_j(s))$.

Caso 2: $e_j+p_j>a_{j+1}$. Programa-se J_{j+1} para começar em e_j+p_j. Nesse caso J_{j+1} possui atraso. A ordem J_{j+1} é inserida no bloco B_l (onde l é o índice do último bloco, isto é, o bloco em análise) e no conjunto *Atrasado*(l).

Uma propriedade-chave, enunciada por Garey *et al.*, mantida pelo algoritmo, é que para cada bloco $B_i \in p_j(s)$, ou $W(i)<H(i)$ ou $e_{primeiro(i)}=0$ (para $i=l=1$). Quando a próxima ordem (J_{j+1}) é programada, apenas o último bloco pode sofrer alguma alteração. Essa alteração pode fazer com que o último bloco tenha pelo menos uma das seguintes propriedades: (a) $W(l)<H(l)$, (b) $W(l)\geq H(l)$ ou (c) $e_{primeiro(l)}=0$.

A segunda parte do algoritmo, de antecipação dos blocos, acontece considerando que se $W(l)<H(l)$ ou $e_{primeiro(l)}=0$, então não se toma nenhuma ação e o programa $p_{j+1}(s)$ transforma-se no programa corrente. Por outro lado, se $W(l)=H(l)$ e $e_{primeiro(l)}\neq 0$, então pode-se movimentar o bloco B_l inteiro, sem afetar o valor da função-objetivo do programa. Caso contrário, se $W(l)>H(l)$ e $e_{primeiro(l)}\neq 0$, pode-se antecipar o bloco B_l inteiro, fazendo com que o valor da função-objetivo do programa diminua. Essas movimentações são feitas até um dos 3 casos seguintes acontecer:

Caso a. $e_{primeiro(l)}=0$. Ocorrerá somente se $l=1$. Caso ocorra, o bloco B_l não pode ser movimentado para mais cedo pois ele estará começando no horário 0;

Caso b. para algum $J_j \in B_l$, $e_j=a_j$. A ordem é transferida do conjunto *Atrasado*(l) para *Adiantado*(l) e $W(l)<H(l)$. Nesse caso, outras movimentações de B_l só irão aumentar o valor da função-objetivo. Caso contrário, se a ordem for transferida de *Atrasado*(l) para *Adiantado*(l) e $W(l)\geq H(l)$, continua-se a movimentação;

Caso c. $e_{primeiro(l)}=e_{último(l-1)}+p_{último(l-1)}$. O bloco B_l é unido ao bloco B_{l-1}. Se $e_{primeiro(l-1)}\neq 0$ e $W(l-1)+W(l)\geq H(l-1)+H(l)$, continua-se a movimentação dos blocos unidos $l-1$ e l. Se $e_{primeiro(l-1)}\neq 0$ e $W(l-1)+W(l)<H(l-1)+H(l)$, deve-se parar a movimentação pois a função-objetivo só irá piorar.

O programa resultante é $p_{j+1}(s)$. O algoritmo faz sucessivas aplicações do procedimento acima formando os programas $p_2(s), p_3(s),...,p_n(s)$.

4.3. ANÁLISE DO ALGORITMO

A demonstração que o algoritmo descrito proporciona um programa ótimo para uma seqüência predefinida no caso sem penalidades é apresentada em Garey *et al.* (1988, p. 338). O teorema adiante prova que o algoritmo descrito aqui garante a otimização do problema em consideração.

Teorema (baseado no teorema 2 de Garey *et al.*). Para qualquer j, o programa parcial $p_j(s)$ calculado pelo algoritmo tem o mínimo $\Sigma(h_jE_j+w_jT_j)$ dentre todos os programas possíveis para as primeiras j ordens.

Demonstração: Ver apêndice. ■

Quando $W(i)=H(i)$ para pelo menos um B_i ($i=1,2,...,l$), podemos perceber que existem infinitos mínimos. Neste caso, o algoritmo proposto irá programar o início do bloco o mais cedo possível.

Deve-se lembrar que a ordem iniciada no horário desejado pertence ao conjunto *Adiantado* e portanto, na melhor condição possível – condição de $g(p)=0$ –, todas as ordens estariam nos conjuntos *Adiantado*. Para o entendimento da propriedade-chave enunciada por Garey *et al.*, ela pode ser dividida em duas partes como segue:

Parte 1: Quando $e_{primeiro(i)}=0$. Só acontece quando $i=l=1$. Não há ociosidade antes do primeiro bloco e conseqüentemente não há como diminuir os atrasos. A relação $W(l)\geq H(l)$ pode ser verdadeira pois não há como movimentar o bloco para mais cedo.

Parte 2: Quando $e_{primeiro(l)}\neq 0$. Para um bloco l com uma ordem apenas, $e_j=a_j$, sendo $primeiro(l)=j$. A ordem J_j pertence ao conjunto $Adiantado(l)$ e $W(l)<H(l)$. Para um bloco l com diversas ordens, vamos aceitar que até a última ordem do mesmo, J_j, a condição $W(l)<H(l)$ seja verdadeira. O acréscimo de J_{j+1} a B_l pode acontecer devido à $a_{j+1}=e_{j+1}$ ou $a_{j+1}<e_{j+1}$, isto é, ou a ordem começa no horário desejado, ou a ordem está atrasada. Quando $a_{j+1}=e_{j+1}$, seguramente a desigualdade $W(l)<H(l)$ se mantém após o acréscimo de J_{j+1} ao bloco B_l, pois J_{j+1} vai para o conjunto $Adiantado(l)$. Quando $a_{j+1}<e_{j+1}$, J_{j+1} vai para o conjunto $Atrasado(l)$ e isso pode nos trazer um dos três resultados a seguir: **(1)** $w_{j+1}+W(l)>H(l)$, a propriedade-chave não é mantida e portanto o bloco deve ser movimentado para mais cedo pois $e_{primeiro(l)}\neq 0$. O bloco é movimentado para mais cedo até que $e_{primeiro(l)}=0$ ou $w_{j+1}+W(l)<H(l)$. Se houver diversos blocos, pode ser necessário haver união dos blocos B_l e B_{l-1} mais de uma vez. As uniões serão realizadas até que pelo menos uma das duas condições de paralização do algoritmo seja alcançada; **(2)** $w_{j+1}+W(l)<H(l)$, a ordem J_{j+1} vai para o conjunto $Atrasado(l)$, mas de maneira contrária ao caso (1), a propriedade-chave é mantida. Uma movimentação para mais cedo fará com que a função-objetivo piore pois a soma das penalidades de adiantamento é maior que a soma das penalidades de atraso; **(3)** $w_{j+1}+W(l)=H(l)$, a ordem J_{j+1} vai para o conjunto $Atrasado(l)$ e as movimentações podem ser feitas pois não irão afetar o valor da função-objetivo. O que se perde quando se adianta as ordens do conjunto $Adiantado(l)$, se ganha com a diminuição do atraso das ordens do conjunto $Atrasado(l)$. A movimentação será feita até que o bloco esteja se iniciando no horário mais cedo possível. O raciocínio precedente pode ser generalizado por indução.

5. IMPLEMENTAÇÃO DO ALGORITMO UTILIZANDO ESTRUTURAS DE FILAS DE PRIORIDADE

Uma implementação eficiente do algoritmo da seção anterior pode ser baseada na estrutura de dados denominada fila de prioridade. Sem a utilização das filas de prioridades, o tempo de solução do algoritmo é $O(n^2)$, enquanto que com a utilização das filas de prioridades é $O(n\log n)$.

De acordo com Tarjan (1983, p. 33), uma fila de prioridade é uma estrutura de dados abstrata consistindo de uma coleção de itens, cada um dos quais possuindo um valor real associado denominado chave.

Para cada bloco de ordens B_i é mantida uma fila de prioridade $P(i)$, que determinará o quanto cada um poderá ser movimentado. Cada fila de prioridade deverá manter dois valores de cada item: o primeiro é o índice das ordens pertencentes a *Atrasado*(i); o segundo é o valor equivalente a $e_j - a_j$ para cada índice j (a quantidade máxima que e_j pode ser diminuída, enquanto diminui o atraso de J_j).

O algoritmo pode ser elaborado considerando-se seis operações com filas de prioridades:

Operação 1. CriaFilaDePrioridade(P). Cria uma fila de prioridade vazia denominada P;

Operação 2. EncontraMin(P). Encontra um item de mínima chave na fila de prioridade P e apresenta essa chave sem remover o item da fila de prioridade;

Operação 3. ApagaMin(P). Encontra um item de mínima chave na fila de prioridade P. Apresenta seu índice e remove o item da fila de prioridade;

Operação 4. Insere(j,x,P). Insere o item j com chave x na fila de prioridade P;

Operação 5. Une(P_1,P_2). Une as filas de prioridades P_1 e P_2 de itens disjuntos em uma nova fila de prioridade que transforma-se em P_1. A fila de prioridade P_2 torna-se vazia;

Operação 6. AdicionaParaTodasChaves(x,P). Adiciona-se x para todas as chaves dos itens pertencentes à fila de prioridade P.

O pseudocódigo utilizado para se inserir a ociosidade em uma seqüência de ordens com os horários de início desejados é apresentado a seguir. Ele é dividido em duas partes, a de construção e a de antecipação de blocos. A Figura 2 mostra a primeira parte do algoritmo enquanto que a Figura 3 mostra a segunda parte do algoritmo.

Após o término do algoritmo, cada ordem $j \in J$ pode ter seu horário efetivo de início definido pela seguinte equação:

$$e_j = \begin{cases} e_{primeiro(i)} \text{ se } j = primeiro(i) \text{ para } i \le l, \\ e_{j-1} + p_{j-1} \text{ caso contrário.} \end{cases}$$

Garey *et al*. e Tarjan discutem tempos de solução computacional, mostrando que as operações com fila de prioridade descritas anteriormente demandam um tempo de no máximo $O(\log n)$ para sua realização. Portanto o algoritmo possui tempo $O(n\log n)$.

```
procedimento Insere Ociosidade (a1,a1,...,an)
início
l:=1; primeiro(1):=1; último(1):=1;
se a1<0 então
    início
    H(1):=0; W(1):=w1; e1:=0;
    fim;
caso contrário
    início
    H(1):=h1; W(1):=0; e1:=a1;
    fim;
CriaFilaDePrioridade(P(1));
para j:=1 até n–1 faça
    se ej+pj<aj+1 então
        início
        l:=l+1; primeiro(l):=j+1; último(l):=j+1; H(l):=hj+1; W(l):=0; ej+1:=aj+1; CriaFilaDePrioridade(P(l));
        fim;
    caso contrário se ej+pj>aj+1 então
        início
        último(l):=j+1; W(l):=W(l)+wj+1; ej+1:=ej+pj; Insere(j+1,ej+1–aj+1,P(l));
        enquanto H(l)≤W(l) faça
            Antecipa;
        fim de enquanto;
        fim;
    caso contrário se ej+pj=aj+1 então
        início
        último(l):=j+1; H(l):=H(l)+hj+1; ej+1:=ej+pj;
        fim;
fim.
```

Figura 2: pseudocódigo para construção de blocos

```
procedimento Antecipa
início
Diferença1:=EncontraMin(P(l));
se l:=1 então
    Diferença2:=e1;
caso contrário
    Diferença2:=eprimeiro(l)–eúltimo(l–1)–púltimo(l–1);
Diferença:=min{Diferença1,Diferença2}; AdicionaParaTodasChaves(–Diferença,P(l));
eprimeiro(l):=eprimeiro(l)–Diferença; eúltimo(l):=eúltimo(l)–Diferença;
enquanto EncontraMin(P(l))=0 faça
    k:=ApagaMin(P(l)); W(l):=W(l)–wk; H(l):=H(l)+ hk;
fim de enquanto;
se l>1 e eprimeiro(l)=eúltimo(l–1)+púltimo(l–1) então
    início
    Une(P(l–1),P(l)); último(l–1):=último(l); H(l–1):=H(l–1)+H(l); W(l–1):=W(l–1)+W(l);
    l:=l–1;
    fim;
fim;
```

Figura 3: pseudocódigo para antecipação dos blocos

6. EXEMPLO

Suponha um conjunto de ordens já seqüenciado, definido pela Tabela 1, com todas ordens disponíveis no instante zero. Queremos saber se a inserção de ociosidade nessa dada seqüência pode promover uma diminuição no valor da função-objetivo.

Por uma questão de comparação, será calculado inicialmente o valor da função-objetivo da seqüência sem a inserção de ociosidade. A Tabela 2 ilustra alguns passos intermediários para esse caso.

j	p_j	d_j	h_j	w_j
1	45	223	50	10
2	82	218	97	58
3	48	285	92	50
4	73	190	35	84
5	36	190	45	2

Tabela 1: Dados do exemplo para inserção de ociosidade

j	C_j	E_j	T_j	$h_jE_j+w_jT_j$
1	45	178	0	8900
2	127	91	0	8827
3	175	110	0	10120
4	248	0	58	4872
5	284	0	94	188
			total	32907

Tabela 2: Cálculo do exemplo sem inserção de ociosidade

O valor da função-objetivo em análise sem a inserção de ociosidade é dado por $\Sigma(h_jE_j+w_jT_j)$=32907. Utilizando-se o algoritmo de inserção de ociosidade, podemos facilmente construir a Tabela 3.

j	$a_j=d_j-p_j$	W_j	C_j	E_j	T_j	$h_jE_j+w_jT_j$
1	178	91	136	87	0	4350
2	136	0	218	0	0	0
3	237	19	285	0	0	0
4	117	0	358	0	168	14112
5	154	0	394	0	204	408
					total	18870

Tabela 3: Cálculo do exemplo com inserção de ociosidade

Os valores intermediários dos horários efetivos de início das ordens obtidos pelo algoritmo são os seguintes: e_1=178; e_1=178, e_2=223; e_1=91, e_2=136; e_1=91, e_2=136, e_3=237; e_1=91, e_2=136, e_3=237, e_4=285; e_1=91, e_2=136, e_3=237, e_4=285, e_5=358. Os blocos existentes após o algoritmo ter sido finalizado são os seguintes: $B_1=\{J_1,J_2\}$ e $B_2=\{J_3,J_4,J_5\}$.

Os blocos indicam que as ordens devem ser produzidas sem a inserção de ociosidade entre as ordens do mesmo bloco. O valor da função-objetivo nesse caso é equivalente a $\Sigma(h_jE_j+w_jT_j)$=18870, que de acordo com o teorema é comprovadamente o mínimo valor alcançado para a seqüência definida inicialmente.

7. APÊNDICE: DEMONSTRAÇÃO DO TEOREMA

Para a demonstração do teorema, por uma questão de simplicidade notacional, será omitida a seqüência na qual o programa está sendo construído, ou seja, o programa parcial $p_j(s)$ será representado por p_j. A representação do valor da função objetivo de um programa parcial até as primeiras j ordens será $g(p_j) = \sum_{k=1}^{j}(h_k E_k + w_k T_k)$ e de um programa completo será $g(p) = \sum_{k=1}^{n}(h_k E_k + w_k T_k)$.

Teorema (baseado no teorema 2 de Garey *et al.* (1988)). Para qualquer j, o programa parcial p_j calculado pelo algoritmo tem o mínimo $g(p_j)$ dentre todos os programas possíveis para as primeiras j ordens.

Demonstração. Como hipótese de indução, assumimos que em j e em $\varnothing$ o teorema é verdadeiro. Portanto, só falta provar que para j+1 o teorema também é verdadeiro. Seja p^*_{j+1} algum programa das primeiras j+1 ordens e seja e_k e e^*_k os horários efetivos de início da ordem J_k em p_j e p^*_j, respectivamente. Devemos considerar dois casos:

Caso sem atraso. Se $e_j + p_j \le a_{j+1}$, então para p_{j+1} não haverá aumento na função-objetivo, $g(p_{j+1}) = g(p_j)$, pois a ordem J_{j+1} é programada para iniciar em a_{j+1}. Também podemos perceber que $E_{j+1} = T_{j+1} = 0$. Mas $g(p^*_{j+1}) \ge g(p_j)$ pois p^*_{j+1} sem a ordem J_{j+1}, é um programa para as primeiras j ordens, tendo a função-objetivo com valor de no mínimo $g(p_j)$ pela hipótese de indução. Como $E^*_{j+1} \ge 0$ e $T^*_{j+1} \ge 0$, podemos concluir que $g(p^*_{j+1}) \ge g(p_{j+1})$ fazendo com que o teorema seja verdadeiro.

Caso com atraso. Se $e_{j+1} > a_{j+1}$, então haverá atraso e portanto aumento no valor da função-objetivo. Seja p_{j+1} o programa definido pelo algoritmo formado por p_j acrescido da ordem J_{j+1} iniciada em $e_{j+1} = e_j + p_j$. Então $g(p_{j+1}) = g(p_j) + w_{j+1}T_{j+1}$. Para a comparação entre $g(p_{j+1})$ e $g(p^*_{j+1})$ podemos considerar dois casos distintos:

Caso 1 com atraso: Se $e^*_{j+1} \ge e_{j+1}$, então $g(p_j) + w_{j+1}(e^*_{j+1} - a_{j+1}) \ge g(p_j) + w_{j+1}(e_{j+1} - a_{j+1})$ o que nos leva a $g(p^*_{j+1}) \ge g(p_{j+1})$, fazendo com que o teorema se mantenha verdadeiro.

Caso 2 com atraso: No último caso $e^*_{j+1} < e_{j+1}$. Podemos considerar que haja 4 hipóteses de alteração do programa p^*_{j+1} com relação ao programa p_{j+1}: (1) não há união de blocos e nenhuma ordem passa do conjunto *Atrasado*(l) para o conjunto *Adiantado*(l); (2) há união de blocos; (3) há passagem de pelo menos uma ordem do conjunto *Atrasado*(l) para o conjunto *Adiantado*(l); (4) há uma combinação das hipóteses anteriores. Na hipótese 1, o último bloco do programa p^*_{j+1}, $B_l \in p^*_{j+1}$, teria todas as ordens adiantadas em uma quantidade equivalente a $e_{j+1} - e^*_{j+1}$. Nesse caso $g(B_l \in p^*_{j+1}) = \sum_{k \in B_l}\{h_k E^*_k + w_k T^*_k\}$, e considerando que $E^*_k = E_k + (e_{j+1} - e^*_{j+1})$ e $T^*_k = T_k - (e_{j+1} - e^*_{j+1})$, podemos dizer que

$$g(B_l \in p^*_{j+1}) = \sum_{k\in B_l}\{h_k E_k + w_k T_k\} + (e_{j+1} - e^*_{j+1})\left\{\sum_{k\in Adiantado(l)} h_k - \sum_{k\in Atrasado(l)} w_k\right\}.$$

Como $\left\{\sum_{k\in Adiantado(l)} h_k - \sum_{k\in Atrasado(l)} w_k\right\} = H(l) - W(l) > 0$, $e_{j+1} - e^*_{j+1} > 0$ e $g(B_l \in p_{j+1}) = \sum_{k\in B_l}\{h_k E_k + w_k T_k\}$, na hipótese 1, $g(p^*_{j+1}) > g(p_{j+1})$ e portanto o teorema é verdadeiro. Como na hipótese 2 há união de blocos, e pela propriedade-chave juntamente com a hipótese de indução sabemos que $H(l) + H(l-1) > W(l) + W(l-1)$, o mesmo raciocínio da hipótese 1 pode ser aplicado e portanto o teorema continua sendo verdadeiro. Na hipótese 3, há passagem de pelo menos uma ordem do conjunto *Atrasado*(l) para o conjunto *Adiantado*(l), mantendo $H(l) > W(l)$ e fazendo com que o raciocínio da hipótese 1 possa ser aplicado, sendo assim o teorema verdadeiro. Finalmente, na hipótese 4 acontece uma combinação dos casos anteriores, fazendo mais uma vez com que o teorema continue sendo verdadeiro. ■

AGRADECIMENTOS

Gostaríamos de agradecer aos revisores, cujas observações e comentários fizeram com que este trabalho tivesse uma melhora significativa. O primeiro autor recebeu apoio financeiro da CAPES e da FAPESP (processo número 97/11282-2) por intermédio de bolsas de mestrado e de doutoramento respectivamente.

REFERÊNCIAS BIBLIOGRÁFICAS

(1) Baker, K. R. & Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties: a review. *Operations Research*, **38**, 22-36.

(2) Colin, E. C. (1997). Beam search e inserção de ociosidade no problema de programação de uma máquina em ambiente do tipo JIT. Dissertação (Mestrado), Universidade de São Paulo, Escola Politécnica.

(3) Fry, T. D., Armstrong, R. D. & Blackstone, J. H. (1987). Minimizing weighted absolute deviation in single machine scheduling. *IIE Transactions*, **19**, 445-450.

(4) Davis, J. S. & Kanet, J. J. (1993). Single-machine scheduling with early and tardy completions costs. *Naval Research Logistics*, **40**, 85-101.

(5) Garey, M. R., Tarjan, R. E. & Wilfong, G. T. (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, **13**, 330-348.

(6) Groenevelt, H. (1993) The just-in-time system. **In**: *Logistics of production and inventory* [edited by S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin], North-Holland, Amsterdam, 629-670.

(7) Morton, T. E. & Pentico, D. W. (1993). *Heuristic scheduling systems: with applications to production systems and project management*. Wiley, New York.

(8) Silver, E. A.; Pyke, D. F. & Peterson, R. (1998). *Inventory management and production planning and scheduling*. 3. ed. Wiley, New York.

(9) Tarjan, R. E. (1983). *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, Philadelphia.

(10) Vollmann, T. E.; Berry, W. L. & Whybark, D. C. (1997). *Manufacturing planning and control systems*. 4. ed. McGraw-Hill, New York.

(11) Yano, C. A. & Kim, Y.-D. (1991). Algorithms for a class of single-machine weighted tardiness and earliness problems. *European Journal of Operational Research*, **52**, 167-178.