UMA HEURÍSTICA CONSTRUTIVA PARA O PROBLEMA DE MINIMIZAÇÃO DO ATRASO TOTAL NO AMBIENTE *FLOWSHOP* COM *BUFFER* ZERO

Luís Roberto Sant'Anna Henriques
Débora P. Ronconi [1]
Escola Politécnica da Universidade de São Paulo
Av. Prof. Almeida Prado, 128, Cidade Universitária, São Paulo. CEP 05508-900
luis.henriques@poli.usp.br
dronconi@usp.br

Resumo
Embora existam muitos trabalhos que tratam do problema de programação de tarefas no ambiente *flowshop* com o objetivo de minimizar o *makespan*, poucos abordam este problema envolvendo datas de entrega. Com o aumento do nível de exigência dos clientes, pesquisas que buscam o atendimento das datas de entrega têm se tornado de extrema importância em ambientes de manufatura. Este trabalho analisa o problema de minimização do atraso total no ambiente *flowshop* com *buffer* zero entre as máquinas. Uma heurística construtiva, baseada no algoritmo NEH, é proposta e avaliada em um grupo de 480 problemas teste. Comparações com trabalhos da literatura mostraram que o método sugerido apresenta um melhor desempenho.
Palavras-chave: programação de tarefas, *flowshop* com bloqueio, atraso.

Abstract:
While there are several research works for the flowshop scheduling problem with the objective of minimizing makespan, few works consider scheduling problems involving due dates. The research involving due dates has become extremely important at production environments, after the competition increasing and the raise of clients' demands in the last years. This paper analyses the minimization of total tardiness in the flowshop with blocking problem. A constructive heuristic based on NEH algorithm is proposed and tested in four hundred eighty problems. Comparative tests show that the proposed heuristic presents good results.
Keywords*: scheduling, flowshop with blocking, tardiness.*

1. Introdução

Desde o trabalho de Johnson em 1954, muitas pesquisas foram feitas a respeito de problemas de programação de tarefas em ambientes *flowshop*, porém, a grande maioria está voltada para o objetivo de minimização do *makespan*, isto é, o instante em que a última tarefa é concluída na última máquina. Uma possível razão para isto pode ser o fato de os problemas ficarem mais difíceis quando objetivos envolvendo datas de entrega são considerados (Kim, 1995). Assim, a literatura a respeito de problemas de minimização de atraso total é restrita se comparada ao *makespan*.

O ambiente *flowshop* é caracterizado pelo processamento de n tarefas em m máquinas, sendo que a *j-ésima* operação de cada tarefa será sempre na máquina j. Desta forma, o número total de alternativas é igual a $(n!)^m$, pois, se não houver restrições específicas, é possível gerar $(n!)$ seqüências em cada uma das m máquinas.

O ambiente *permutation flowshop* é uma particularização do ambiente *flowshop*. Nele, existe a restrição de que todas as máquinas processam as tarefas na mesma seqüência, isto é, se uma tarefa k precede uma tarefa u na máquina j, ela será processada antes em qualquer outra máquina. Desta forma, o número total de alternativas é reduzido para $(n!)$, visto que existe uma seqüência única para todas as m máquinas. Apesar desta simplificação, o problema ainda é bastante complexo, sendo *NP-hard* para o

[1] Esta pesquisa possui suporte do Conselho Nacional de Desenvolvimento Científico – CNPq e Fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP (Processo: 01/09012-4)

critério de minimização do atraso total (Kim, 1995). Embora o ambiente de *permutation flowshop* nem sempre contenha o ótimo global, ele não pode ser subestimado, pois, em muitos ambientes reais, especialmente aqueles que possuem esteiras transportadoras, este é o único ambiente viável (Grabowski e Pempera, 2000).

Uma outra particularização do ambiente *flowshop* mais genérico pode ocorrer devido à inexistência de estoques intermediários entre as máquinas, também chamado de ambiente *flowshop* com *buffer* zero ou *flowshop* com bloqueio. Nestes ambientes, formações de filas entre as máquinas não são permitidas. Assim, se uma tarefa *k* estiver concluída na máquina *j* e sua predecessora ainda estiver em processamento na máquina *j+1*, a tarefa *k* bloqueará a máquina *j* até a máquina *j+1* ser liberada. Problemas em ambiente *flowshop* com *buffer* zero sempre são do tipo *permutation flowshop* porque a inexistência de estoques intermediários impede que uma tarefa ainda não processada passe à frente de outra tarefa em processo ou bloqueada.

A razão para o bloqueio das tarefas e da inexistência de estoques intermediários está, muitas vezes, no próprio processo produtivo. Estes ambientes podem ser encontrados, por exemplo, na produção de blocos de concreto (Grabowski e Pempera, 2000). A importância dos problemas com bloqueio é reforçada por McCormick *et al.* (1989), pois, segundo os autores, o ambiente *flowshop* com *buffer* zero pode modelar qualquer outro *flowshop* com restrição de capacidades intermediárias através da inserção de máquinas fictícias com tempos de processamento iguais a zero para representar os pontos de armazenagem.

A pesquisa na área de programação de tarefas é muito rica, porém, a grande maioria dos trabalhos está voltada para o objetivo de minimização do *makespan*. Critérios de otimização envolvendo datas de entrega vêm assumindo uma importância cada vez maior, pois o não cumprimento da data pode gerar uma série de problemas como: penalidades contratuais, perda de credibilidade – o que pode resultar na perda do cliente – e danos à imagem da empresa – podendo afastar novos clientes (Sen e Gupta, 1984). Um outro fator de destaque na literatura de programação de tarefas é o fato de a maioria dos artigos não tratar de problemas com limitações de estoques intermediários (Hall e Sriskandarajah, 1996).

Koulamas (1994) apresenta uma extensa revisão bibliográfica para problemas de programação de tarefas envolvendo critérios relacionados a datas de entrega. Uma revisão para estes problemas no ambiente *flowshop* para a minimização do atraso total sem restrições de estoques intermediários foi apresentada por Kim (1993). Neste artigo, o autor apresenta uma série de heurísticas e constata, através de experimentos computacionais que uma adaptação do algoritmo NEH, proposto por Nawaz *et al*. (1983), partindo da regra EDD apresenta bons resultados em tempos razoáveis.

Além das pesquisas com heurísticas, outros trabalhos envolvendo algoritmos exatos com objetivos relacionados às datas de entrega foram feitos para o ambiente *flowshop*. Kim (1995) apresenta uma série de propriedades para o cálculo de limitantes e propõe a utilização do algoritmo *branch-and-bound* com base nestes limitantes. Além disso, o artigo também apresenta regras de dominância que permitem a exclusão de algumas tarefas. Ainda na área de pesquisa de algoritmos exatos, Grabowski *et al*. (1983) propõem a utilização do *branch-and-bound* para o problema de minimização do atraso máximo no qual os limitantes são obtidos através de relaxamentos nas restrições das capacidades das máquinas.

Com relação aos ambientes com restrição de estoques intermediários, McCormick *et al*. (1989) propõem uma heurística construtiva que utiliza propriedades relacionadas ao caminho crítico para o problema de programação de tarefas com bloqueio para o critério de minimização do ciclo total. Leisten (1990) apresenta heurísticas para o problema nos ambientes *flowshop* com e sem limitações de *buffer* cujos objetivos são maximizar a ocupação dos *buffers* e minimizar os bloqueios das máquinas. Nowicki (1999) sugere a utilização da metaheurística busca tabu para o critério de minimização do *makespan* no ambiente *flowshop* com *buffers* limitados. Além disso, o autor propõe uma generalização das propriedades do bloqueio, propostas originalmente por Grabowski (1983), conhecidas para o *flowshop* clássico. Caraffa *et al.* (2001) propõem a utilização de algoritmos genéticos para a minimização do *makespan* no ambiente *flowshop* com bloqueios. Ainda em problemas para a minimização do *makespan*, Ronconi (2004) propõe heurísticas construtivas para problemas no ambiente *flowshop* com *buffer* zero. Segundo a autora, a utilização de heurísticas construtivas pode ser

motivada e justificada pelo pequeno esforço computacional. Além disso, heurísticas construtivas são geralmente mais simples e fáceis de implementar se comparadas às metaheurísticas e, normalmente, exigem menor esforço para o ajuste de seus parâmetros.

Pesquisas envolvendo algoritmos exatos também foram feitas para ambientes com restrição de estoques intermediários. Levner (1969) propõe a utilização do algoritmo *branch-and-bound* para o problema no ambiente *flowshop* com bloqueios e demonstra que calcular o *makespan* neste ambiente é equivalente a encontrar o maior caminho em um grafo. Suhami e Mah (1981) propõem a utilização do *branch-and-bound* no qual é possível determinar a qualidade da solução através de um parâmetro empírico utilizado no cálculo do limitante inferior.

Especificamente para o ambiente estudado neste trabalho, minimização do atraso total considerando *buffer* zero, Ronconi e Armentano (2001) propõem um limitante inferior para os instantes de término das tarefas e utilizam este limitante na elaboração de um algoritmo *branch-and-bound*. Em um outro trabalho, Armentano e Ronconi (2000) propõem a utilização da metaheurística busca tabu utilizando como solução inicial uma adaptação do algoritmo NEH, chamada de LBNEH, que procura programar as tarefas utilizando as datas de entrega das tarefas como critério principal. A proposta do trabalho atual é contribuir com o estudo deste problema através da elaboração de uma heurística que explora, além de características relacionadas às datas de entrega, características da ausência de *buffers* intermediários entre as máquinas. Os resultados da heurística LBNEH foram utilizados como referência para os resultados obtidos neste trabalho.

A próxima seção contém a descrição do problema. A seção 3 apresenta a heurística construtiva proposta para o problema em questão. Na quarta seção são apresentados os resultados computacionais obtidos com a heurística proposta e a última seção apresenta um resumo dos resultados obtidos neste trabalho.

2. Descrição do Problema

Este trabalho tem como objetivo analisar e propor uma heurística construtiva para o problema de minimização do atraso total no ambiente *flowshop* com *buffer* zero entre as máquinas. Neste problema, os tempos de processamento de cada tarefa em cada máquina e as datas de entrega de cada tarefa são previamente conhecidos e a medida de desempenho a ser avaliada será o atraso total de todas as tarefas. Segundo Du e Leung (1990), o problema de minimização do atraso total para o caso particular de uma máquina é *NP-hard*.

Este problema pode ser classificado pela notação $\alpha|\beta|\gamma$, onde α representa a configuração das máquinas, β representa características especiais do problema e γ representa o objetivo do problema. Desta forma, o problema pode ser descrito por $F_m \mid blocking \mid \Sigma\ T_k$ onde T_k é o atraso da tarefa k – *tardiness*.

Para a avaliação deste problema, foram consideradas as seguintes hipóteses: todas as tarefas estão disponíveis para processamento no instante zero, os tempos de *setup* são independentes da seqüência e estão embutidos nos tempos de processamento, as máquinas estão continuamente disponíveis e não são permitidas interrupções no processamento.

O problema pode ser avaliado utilizando-se as fórmulas apresentadas em Pinedo (1995). Assume-se que $t_1, t_2, \ldots, t_i, \ldots, t_n$ seja uma seqüência proposta para a avaliação das tarefas, onde t_i representa a tarefa k que ocupa a *i-ésima* posição na seqüência e que $M_1, M_2, \ldots, M_j, \ldots, M_m$ seja a seqüência de máquinas nas quais as tarefas deverão ser processadas. Sejam p_{kj} o tempo de processamento da tarefa k na máquina M_j, d_k a data de entrega da tarefa k, $D_{ti,0}$ o instante em que a tarefa t_i começou a ser processada na primeira máquina e $D_{ti,j}$ o instante em que a tarefa t_i partiu da máquina M_j. Os instantes de partida das máquinas são calculados por:

$$D_{t1,0} = 0$$

$$D_{t1,j} = \sum_{q=1}^{j} p_{t1,q} \qquad j = 1,\ldots,m-1$$

$$D_{ti,0} = D_{t(i-1),1} \qquad i = 2,\ldots,n$$

$$D_{ti,j} = \max\left(D_{ti,j-1} + p_{ti,j}, D_{t(i-1),j+1}\right) \qquad i = 2,...,n \qquad j = 1,...,m-1$$
$$D_{ti,m} = D_{ti,m-1} + p_{ti,m} \qquad i = 1,...,n$$

Uma vez conhecido o instante em que cada tarefa foi concluída, é possível calcular o atraso total, dado por:

$$T = \sum_{i=1}^{n} \max\left(D_{ti,m} - d_{ti}, 0\right)$$

3. Heurística Proposta

A heurística proposta é uma adaptação do algoritmo NEH (Nawaz *et al.*, 1983) e gera a solução em duas fases: na primeira fase, as tarefas são ordenadas por uma regra de despacho que considera a possível existência de um bloqueio e, na segunda fase, são seguidos os mesmos passos do esquema de inserção do algoritmo NEH, avaliando as alternativas de acordo com a medida de desempenho do problema.

Diversos testes foram feitos, tanto na primeira como na segunda, fase com o objetivo de selecionar a melhor estratégia heurística. A seguir, são descritos os métodos que obtiveram melhores resultados em conjunto.

3.1. Primeira fase

O algoritmo PERFDAT (perfil de datas) procura gerar uma solução inicial para o problema buscando organizá-las respeitando o perfil dos tempos de processamento e das datas de entrega. A técnica trabalha de forma dinâmica, isto é, após a seleção de uma tarefa para ocupar um lugar na seqüência já fixada (σ), são recalculados os fatores de prioridade de todas as outras tarefas não pertencentes à seqüência σ antes de se fazer nova escolha.

Observou-se que a escolha da tarefa que ocupa o primeiro lugar na seqüência é de grande impacto no desempenho do método, principalmente porque tarefas com tempo de processamento muito alto na primeira máquina podem acarretar um baixo desempenho do sistema. Isto ocorre porque enquanto esta está sendo processada na primeira máquina, todas as outras tarefas estão aguardando, não havendo nenhum outro processamento. Várias alternativas foram avaliadas para a seleção desta primeira tarefa, dentre estas, a que apresentou melhores resultados foi escolher a tarefa que apresentasse o menor fator I_k, dado por: $I_k = d_k + p_{k1}$.

Para a escolha das tarefas seguintes, procurou-se respeitar o perfil dos tempos de processamento da última tarefa (t_i) fixada na seqüência σ, além das datas de entregas das tarefas consideradas. Calcula-se, para cada tarefa $k \notin \sigma$, o fator de prioridade dado por $F_k = \sum_{j=1}^{m-1} \left| b_j - p_{kj} \right| + \left(LB_k - D_{ti,1}\right)$, onde:

- b_j é o intervalo de tempo disponível em cada uma das máquinas, dado por: $b_j = D_{ti,j+1} - D_{ti,j}$, para $j = 1, 2, ..., m-1$.
- LB_k é a "sobra" da tarefa k (Armentano e Ronconi, 1999), dada por: $LB_k = d_k - \sum_{j=1}^{m} p_{kj}$.

O primeiro termo do fator de prioridade F_k dá preferência para a tarefa que possui o perfil de tempos de processamento mais próximo das janelas disponíveis (b_j), sem que a mesma provoque um bloqueio ou fique esperando a liberação da máquina seguinte. O segundo termo procura aumentar a importância das tarefas cujos espaços de tempo para serem processadas estão reduzidos. Deve-se escolher a tarefa que apresente o menor fator F_k.

A Figura 1 apresenta um exemplo com cinco máquinas e com três tarefas já fixadas na seqüência σ.

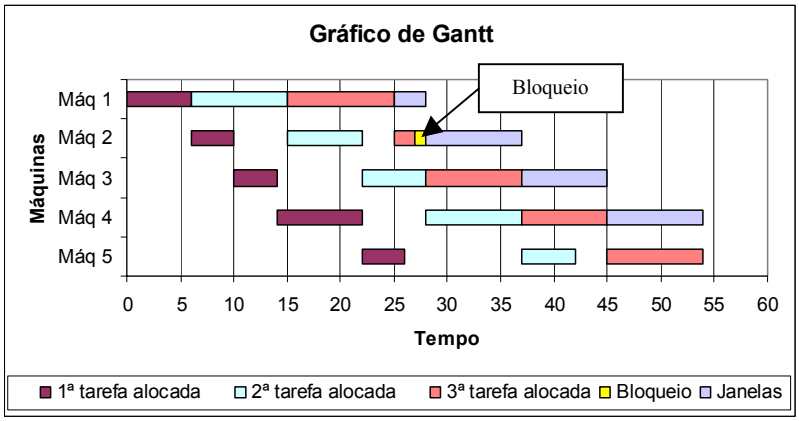

Figura 1: Representação gráfica das janelas.

A seguir, é apresentada uma breve descrição de como algoritmo PERFDAT ordena as tarefas:

Passo 1. Iniciar a seqüência $\sigma = \varnothing$.

Passo 2. Calcular a "sobra" (LB_k) de todas as tarefas.

Passo 3. Calcular o fator I_k para todas as tarefas e escolher aquela que apresentar o menor valor para ocupar o primeiro lugar da seqüência σ.

Passo 4. Calcular os instantes de partida ($D_{ti,j}$) da última tarefa alocada em cada uma das máquinas, onde i é a posição da última tarefa seqüenciada em σ.

Passo 5. Calcular as janelas disponíveis em cada uma das máquinas (b_j).

Passo 6. Calcular o fator F_k, para todas as tarefas ainda não alocadas, ou seja, para $k \notin \sigma$.

Passo 7. Colocar a tarefa com menor fator F_k no próximo lugar da seqüência σ.

Passo 8. Repetir os passos 4, 5, 6 e 7 até que todas as tarefas tenham sido alocadas.

A tabela 1 apresenta os dados de um exemplo com 4 tarefas e 5 máquinas para ilustrar o algoritmo proposto:

Tarefa	Tempos de Processamento					$\Sigma\, p_{kj}$	d_k	LB_k	$I_k = d_k + p_{k1}$
	M_1	M_2	M_3	M_4	M_5				
1	6	4	4	8	4	26	29	3	35
2	10	2	9	8	9	38	45	7	55
3	9	7	6	9	5	36	40	4	49
4	6	7	8	1	7	29	43	14	49

Tabela 1: Dados do exemplo do algoritmo PERFDAT.

A tarefa 1 foi escolhida para a 1ª posição aplicando-se os 3 primeiros passos. Assim, calculam-se os instantes de partida e as janelas disponíveis para cada uma das máquinas (Passos 4 e 5 – Tabela 2).

Tarefa 1	$D_{ti,j-1}$	$D_{ti,j}$	b_j
M_1	0	6	4
M_2	6	10	4
M_3	10	14	8
M_4	14	22	4
M_5	22	26	-

Tabela 2: Algoritmo PERFDAT, 1ª iteração (cálculo de b_j).

Calcula-se, segundo o passo 6, o fator F_k para as tarefas restantes (Tabela 3).

Tarefa	$\|b_j - p_{kj}\|$				$LB_k - D_{ti,1}$	F_k
	M_1	M_2	M_3	M_4		
2	6	2	1	4	1	14
3	5	3	2	5	-2	13
4	2	3	0	3	8	16

Tabela 3: Algoritmo PERFDAT, 1ª iteração (cálculo F_k).

A tarefa 3 foi escolhida para a 2ª posição (Passo 7). Assim, os instantes de partida e as janelas disponíveis para cada uma das máquinas são calculados (Passos 4 e 5 – Tabela 4).

Tarefa 3	$D_{ti,j-1}$	$D_{ti,j}$	b_j
M_1	6	15	7
M_2	15	22	6
M_3	22	28	9
M_4	28	37	5
M_5	37	42	-

Tabela 4: Algoritmo PERFDAT, 2ª iteração (cálculo de b_j).

Calcula-se, segundo o passo 6, o fator F_k para as tarefas restantes (Tabela 5).

Tarefa	$\|b_j - p_{kj}\|$				$LB_k - D_{ti,1}$	F_k
	M_1	M_2	M_3	M_4		
2	3	4	0	3	-8	2
4	1	1	1	4	-1	6

Tabela 5: Algoritmo PERFDAT, 2ª iteração (cálculo F_k).

A tarefa 2 foi escolhida para a 3ª posição (Passo 7). Assim, os instantes de partida e as janelas disponíveis para cada uma das máquinas são calculados (Passos 4 e 5 – Tabela 6).

Tarefa 2	$D_{ti,j-1}$	$D_{ti,j}$	b_j
M_1	15	25	3
M_2	25	28	9
M_3	28	37	8
M_4	37	45	9
M_5	45	54	-

Tabela 6: Algoritmo PERFDAT, 3ª iteração (cálculo de b_j).

Calcula-se, segundo o passo 6, o fator F_k para as tarefas restantes (Tabela 7).

Tarefa	$\|b_j - p_{kj}\|$				$LB_k - D_{ti,1}$	F_k
	M_1	M_2	M_3	M_4		
4	3	2	0	8	0	13

Tabela 7: Algoritmo PERFDAT, 3ª iteração (cálculo F_k).

A tarefa 4 foi escolhida para a 4ª posição (Passo 7). A solução inicial possui a seguinte seqüência: 1, 3, 2 e 4, com um atraso total de 29. Pode-se perceber pelo gráfico de Gantt da solução inicial, apresentado na Figura 2, que o algoritmo PERFDAT gerou uma solução com bloqueios de algumas tarefas, porém, esses bloqueios não são muito freqüentes.

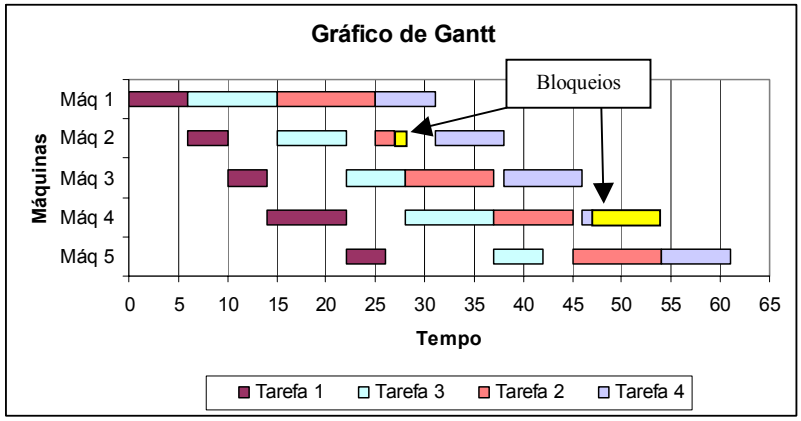

Figura 2: Gráfico de Gantt da solução obtida pelo algoritmo *PERFDAT*.

3.2. Segunda fase

Uma vez classificadas pelo algoritmo PERFDAT, as tarefas são reordenadas pelo algoritmo NEHPERF que, diferentemente do algoritmo NEH original proposto para minimizar o *makespan*, avalia as alternativas buscando minimizar a somatória dos atrasos.

As duas primeiras tarefas da seqüência obtida pelo algoritmo PERFDAT são selecionadas, as duas seqüências possíveis para elas são geradas e avaliadas de acordo com a somatória dos atrasos e a seqüência com melhor resultado parcial é escolhida. Uma vez feito isso, a posição relativa entre estas duas tarefas é fixada. Em seguida, seleciona-se a terceira tarefa da seqüência obtida pelo algoritmo PERFDAT e as três seqüências possíveis, posicionando a tarefa no início, no meio e no fim da seqüência anteriormente obtida, são avaliadas. As posições serão fixadas de acordo com a melhor solução parcial. Este processo é repetido até que todas as tarefas estejam fixadas, quando a seqüência final é encontrada.

O exemplo a seguir ilustra esta segunda fase do problema:

Seqüência inicial (PERFDAT): 1, 3, 2 e 4. Atraso total: 29.

Escolhem-se as duas primeiras tarefas e avaliam-se as duas possíveis seqüências:

Seqüência: 1 e 3. Atraso total: 2.

Seqüência: 3 e 1. Atraso total: 14.

A seqüência 1, 3 foi escolhida. Seleciona-se a terceira tarefa da solução inicial (tarefa 2) e avaliam-se as três possíveis seqüências.

Seqüência: 2, 1 e 3. Atraso total: 25.

Seqüência: 1, 2 e 3. Atraso total: 12.

Seqüência: 1, 3 e 2. Atraso total: 11.

A seqüência 1, 3, 2 foi escolhida. Seleciona-se a quarta tarefa da solução inicial (tarefa 4) e avaliam-se as quatro possíveis seqüências.

Seqüência: 4, 1, 3 e 2. Atraso total: 33.

Seqüência: 1, 4, 3 e 2. Atraso total: 23.

Seqüência: 1, 3, 4 e 2. Atraso total: 26.

Seqüência: 1, 3, 2 e 4. Atraso total: 29.

Após a execução do algoritmo NEHPERF a seqüência encontrada foi 1, 4, 3 e 2 com atraso total de 23. O gráfico de Gantt da solução é apresentado na Figura 3:

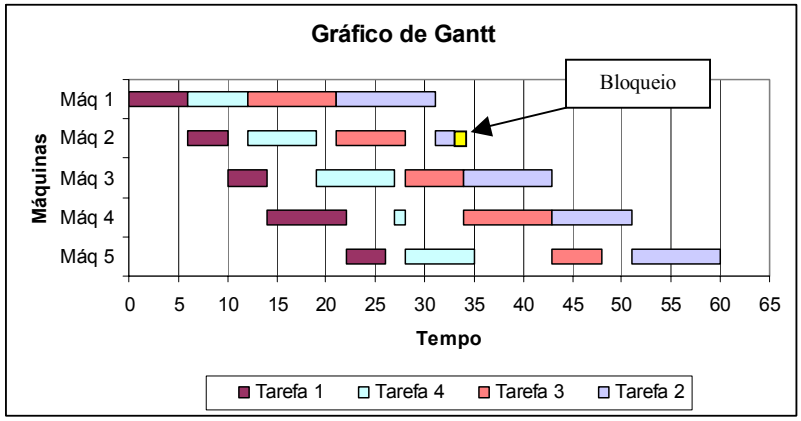

Figura 3: Gráfico de Gantt da solução obtida pelo algoritmo *NEHPERF*.

Pode-se perceber que após a execução da segunda fase houve uma melhoria na qualidade da solução encontrada além de haver uma redução do número de bloqueios. Neste caso, a solução ótima foi encontrada.

4. Resultados Computacionais

Os métodos propostos foram implementados em linguagem de programação C++ e os testes foram realizados em um microcomputador Pentium IV com processador de 2,3GHz e 512Mb de memória RAM. Os problemas teste foram gerados conforme Taillard (1993). Para cada dimensão (Tabela 10) foram analisadas dez matrizes de tempos de processamento e, para cada caso, foram gerados quatro cenários de datas de entrega.

As datas de entrega foram geradas uniformemente dentro do intervalo $\left[P\left(1-T-R/2\right), P\left(1-T+R/2\right)\right]$, onde T e R representam, respectivamente, fator de atraso e amplitude, e P é um limitante inferior do *makespan* para o problema no ambiente *flowshop* sem limitação de estoques intermediários, definido por:

$$P = \max\left\{\max_{1\le v\le m}\left[\sum_{k=1}^{n} p_{kv} + \min_{k}\sum_{q=1}^{v-1} p_{kq} + \min_{k}\sum_{q=v+1}^{m} p_{kq}\right], \max_{k}\sum_{v=1}^{m} p_{kv}\right\}$$

Os quatros cenários de datas de entrega foram gerados por variações dos parâmetros T e R, conforme descrito a seguir:

Cenário 1: baixo fator de atraso (T=0,2) e pequena faixa de datas de entrega (R=0,6).

Cenário 2: baixo fator de atraso (T=0,2) e ampla faixa de datas de entrega (R=1,2).

Cenário 3: alto fator de atraso (T=0,4) e pequena faixa de datas de entrega (R=0,6).

Cenário 4: alto fator de atraso (T=0,4) e ampla faixa de datas de entrega (R=1,2).

Assim, foram avaliados 480 problemas e a avaliação da heurística construtiva foi feita de forma comparativa, considerando-se a melhoria obtida em relação à heurística LBNEH, proposta por Armentano e Ronconi (2000). A medida da melhoria é calculada através da seguinte expressão: $Melhoria = \frac{A_{LBNEH} - A_{NEHPERF}}{A_{LBNEH}}$, onde A_{LBNEH} e $A_{NEHPERF}$ representam o atraso total obtido por cada um dos algoritmos.

4.1. Testes preliminares

Diversas variações, tanto na primeira como na segunda fase, com o objetivo de melhorar o desempenho da heurística construtiva foram feitas. Na primeira fase, foram propostas variações no critério de escolha da primeira tarefa (fator I_k) e no critério de escolha das outras tarefas (fator F_k). A tabela 8 apresenta as diferentes versões avaliadas.

Fator	Forma de Cálculo	Versão
I	$I_k = d_k + p_{k1}$	A
I	$I_k = d_k$	B
I	$I_k = LB_k$	C
F	$F_k = \sum_{j=1}^{m-1} \left\| b_j - p_{kj} \right\| + \left(LB_k - D_{ti,1} \right)$	α
F	$F_k = \sum_{j=1}^{m-1} \left\| b_j - p_{kj} \right\| + \max\left(LB_k - D_{ti,1} \right)$	β

Tabela 8: Testes para a primeira fase.

Na segunda fase, foram propostas algumas variações para lidar com o grande número de empates nas soluções parciais encontradas pelo algoritmo NEH adaptado (Tabela 9), especialmente na fixação das primeiras posições.

Versão	Descrição
1	No caso de haver um empate, o algoritmo mantém a primeira seqüência parcial encontrada.
2	No caso de haver um empate, o algoritmo compara os tempos de processamento da tarefa na primeira posição em M_1 das seqüências candidatas e seleciona a seqüência que possuir menor tempo. No caso de empate o algoritmo compara as soma dos tempos de processamento em M_1 das duas primeiras tarefas e seleciona a seqüência com menor soma. Este processo é repetido para as outras tarefas enquanto houver empates.
3	O algoritmo NEH adaptado inicia a fixação das tarefas pelas últimas posições. Assim, foi necessário o cálculo de um limitante inferior (LI) de liberação de cada máquina. Utilizamos o LI sugerido por Kim (1993).
4	Idéia semelhante à anterior, utilizando-se de um LI mais aderente ao problema, sugerido por Ronconi e Armentano (2001).

Tabela 9: Testes para a segunda fase.

A versão "Aα1" foi a que obteve melhores resultados provavelmente porque, ao escolher a primeira tarefa da seqüência, penaliza aquelas que possuem tempo de processamento bastante alto na primeira máquina, o que geraria espera para todas as outras tarefas. Além disso, ela prioriza aquelas tarefas que já não possuem tempo suficiente para o seu processamento sem causar atraso. No caso de haver empate entre duas ou mais soluções, o critério que apresentou melhores resultados foi manter a primeira solução encontrada provavelmente porque esta, devido a características da implementação, é mais "próxima" à solução encontrada pelo algoritmo PERFDAT, que buscou gerar uma solução com o menor número de bloqueios.

4.2. Comparação com o algoritmo LBNEH

A tabela 10 apresenta a melhoria percentual média do NEHPERF em relação ao LBNEH. Os valores destacados em negrito indicam os grupos de problemas nos quais o algoritmo NEHPERF obteve uma melhoria média superior a 10% em relação ao algoritmo LBNEH enquanto os valores negativos indicam os grupos de problemas nos quais, em média, não houve melhoria.

Dimensão	Cenário				Média por
(*n* x *m*)	1	2	3	4	dimensão
20x5	**11,62**	5,28	-0,04	0,76	4,40
20x10	2,28	5,05	0,25	0,71	2,07
20x20	0,86	-1,72	0,14	-0,19	-0,23
50x5	6,14	**33,09**	7,87	3,53	**12,66**
50x10	1,91	8,35	-0,71	0,88	2,61
50x20	2,72	-2,34	-0,51	-1,80	-0,48
100x5	**16,52**	**24,44**	4,47	0,45	**11,47**
100x10	9,91	9,10	2,63	1,03	5,67
100x20	4,74	7,37	0,46	0,13	3,18
200x10	**10,06**	**14,09**	3,81	0,99	7,24
200x20	6,02	6,63	2,37	0,17	3,80
500x20	7,35	8,17	2,51	0,21	4,56
Média	6,68	9,79	1,94	0,57	4,75

Tabela 10: Melhoria percentual do NEHPERF em relação ao LBNEH.

Em 85,4% das classes avaliadas (Dimensão x Cenário), o algoritmo NEHPERF superou o algoritmo LBNEH, sendo que, em 13% das classes avaliadas, a melhoria média foi superior a 10%. Isto, provavelmente, se deve ao fato do NEHPERF considerar mais características inerentes ao problema ao gerar a solução da primeira fase. Através da análise desta tabela, é possível perceber que o algoritmo NEHPERF obteve melhores resultados nos cenários com baixo fator de atraso. Além disso, observa-se que o aumento do número de tarefas, mantendo-se o número de máquinas constante, também favorece o algoritmo NEHPERF na maioria dos casos.

A tabela 11 apresenta uma nova comparação entre os algoritmos LBNEH e NEHPERF. Nota-se que, em 332 dos 480 problemas, houve melhoria em relação ao algoritmo LBNEH e que, no melhor caso, houve um ganho de 68,05% na qualidade da solução. Além disso, somente em 13 problemas a perda foi superior a 10%, ao passo que, em 102, o ganho foi superior a 10%.

Melhoria percentual em relação ao LBNEH	**Pior caso**	-39,37%
	Melhor caso	68,05%
	Média	4,75%
	Melhoria > 10%	102
	0 < Melhoria ≤ 10%	230
	Melhoria = 0	13
	-10% ≤ Melhoria < 0	122
	Melhoria < -10%	13

Tabela 11: Comparação do NEHPERF em relação ao LBNEH.

5. Conclusões

O objetivo deste trabalho foi propor uma heurística construtiva para o problema de minimização do atraso total em ambiente *flowshop* com buffer zero considerando algumas características específicas do problema como, por exemplo, o possível bloqueio das máquinas. Observou-se que a introdução destas características apresentou melhorias significativas nos resultados sem grande aumento na complexidade da heurística.

Dentre todas as versões avaliadas para a heurística construtiva, somente aquela que apresentou melhor desempenho foi apresentada de forma detalhada neste artigo. Porém, a grande maioria delas apresentou bons resultados e, somente as versões que utilizaram critérios com limitantes na segunda fase não superaram a heurística LBNEH, utilizada como referência. Como futuras pesquisas, sugere-se a aplicação de metaheurísticas com intuito de melhorar a qualidade das soluções obtidas neste estudo.

Bibliografia

ARMENTANO, V. A., RONCONI, D. P. (2000) – Minimização do tempo total de atraso no problema de flowshop com buffer zero através de busca tabu – *Gestão & Produção*, Vol. 7, No. 3, pp. 352 – 363.

ARMENTANO, V. A., RONCONI, D. P. (1999) – Tabu search for total tardiness minimization in flowshop scheduling problems – Computers & Operations Research, Vol. 26, pp. 219 – 235.

CARAFFA, V., IANES, S., BAGCHI, T. P., SKISKANDARAJAH, C. (2001) – Minimizing makespan in a blocking flowshop using genetic algorithms – *International Journal of Production Economics*, Vol. 70, pp. 101 – 115.

DU, J., LEUNG JYT. (1990) – Minimizing total tardiness on one machine is NP-hard – *Mathematics of Operations Research*, Vol. 15, pp. 483-95.

GRABOWSKI, J., SKUBALSKA, E., SMUTNICKI, C. (1983) – On Flow Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness – *Journal of the Operational Research Society*, Vol. 34, No. 7, pp. 615 – 620.

GRABOWSKI, J., PEMPERA, J. (2000) – Sequencing of jobs in some production system – *European Journal of Operational Research*, Vol. 125, pp. 535 – 550.

HALL, N. G., SRISKANDARAJAH, C. (1996) – A survey of machine scheduling problems with blocking and no-wait in process – *Operations Research*, Vol. 44, No. 3, 510-525.

JOHNSON, S. M. (1954) – Optimal two and three stage production schedules with setup times included – *Naval Research Logistics Quarterly*, Vol. 1, pp. 61 – 67.

KIM, Y. D. (1993) – Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness – *Journal of the Operational Research Society*, Vol. 44, No. 1, pp. 19 – 18.

KIM, Y. D. (1995) – Minimizing total tardiness in permutation flowshops – *European Journal of Operational Research*, Vol. 85, pp. 541 – 555.

KOULAMAS, C. (1994) – The total tardiness problem: review and extensions – *Operations Research*, Vol. 42, No. 6, pp. 1025 – 1041.

LEISTEN, R. (1990) – Flowshop sequencing problems with limited buffer storage – *International Journal of Production Research*, Vol. 28, No. 11, pp. 2085 – 2100.

LEVNER, E. V. (1969) – Optimal planning of parts machining on a number of machines – *Automation and Remote Control*, No. 12, pp. 1972 – 1978.

MCCORMICK, S. T., PINEDO, M. L., SHENKER, S., WOLF, B. (1989) – Sequencing in an assembly line with blocking to minimize cycle time – *Operations Research*, Vol. 37, No. 6, pp. 925 – 935.

NAWAZ, M., ENSCORE, E., HAM, I. (1983) – A Heuristic Algorithm for the m-Machine n-Job Flow-shop Sequencing Problem – *The International Journal of Management Science*, Vol. 11, No. 1, pp. 91 – 95.

NOWICKI, E. (1999) – The permutation flow shop with buffers: A tabu search approach – *European Journal of Operational Research*, Vol. 116, pp. 205 – 219.

PINEDO, M. (1995) – *Scheduling: Theory Algorithms and Systems* – Prentice-Hall, Englewood Cliffs, NJ.

RONCONI, D. P. (2004) – A note on constructive heuristics for the flowshop problem with blocking – International Journal of Production Economics, Vol. 87, pp. 39 – 48.

RONCONI, D. P., ARMENTANO, V. A. (2001) – Lower bounding schemes for flowshop with blocking in-process – *Journal of the Operational Research Society*, Vol. 52, pp. 1289 – 1297.

SEN, T., GUPTA, S. K. (1984) – A state-of-art survey of scheduling research involving due dates – *OMEGA*, Vol. 12, pp. 63 – 76.

SUHAMI, I., MAH, R. S. H. (1981) – An implicit enumeration scheme for the flowshop problem with no intermediate storage – *Computers and Chemical Engineering*, No. 5, pp. 83 – 91.

TAILLARD, E. (1993) – Benchmarks for basic scheduling problems – *European Journal of Operational Research*, Vol. 64, pp. 278 – 285.