O ALGORITMO GENÉTICO PARA SOLUCIONAR A PROGRAMAÇÃO DE PRODUÇÃO DO SISTEMA DE FLUXO PERMUTACIONAL

THE GENETIC ALGORITHM TO SOLVING A FLOW SHOP PERMUTATION SCHEDULING

Hieronim Napierala[1]

RESUMO: Este trabalho objetiva apresentar o algoritmo evolutivo híbrido (AEH) para solucionar o problema de programação das operações no ambiente de *flow shop permutation,* que minimiza o tempo máximo do término das operações. A nova heurística baseia-se no algoritmo genético que incluí a busca local. Foram desenvolvidos testes computacionais e avaliado o método proposto. A qualidade da solução é notável e o desempenho aceitável, se comparado aos resultados encontrados na bibliografia da pesquisa operacional sobre tal problema.

Palavras-chave: *flow shop* permutacional, *makespan*, metaheurística.

ABSTRACT: This study presents a hybrid evolutionary algorithm (AEH) to solve the permutation flow shop scheduling problem, with the objective of minimizing the maximum time of termination – makespan. The new heuristics based on genetic algorithm that includes a local search. We developed the computational tests and the proposed method was evaluated. The solution quality is outstanding and acceptable performance compared with the results found in the bibliography of operation research about this problem.

Keywords: permutation flow shop, makespan, metaheuristic.

1 INTRODUÇÃO

O ambiente atual no qual as empresas realizam os seus negócios caracteriza se pela demanda variável, falta de fidelidade dos clientes, diminuição do ciclo de vida de produtos e por uma forte e global concorrência. Para darem continuidade aos seus negócios, as empresas necessitam de uma estrutura de informação e de

[1] Doutor em Engenharia de Produção pela Universidade Federal de Santa Catarina. Atualmente é Professor Associado da Universidade Estadual do Oeste do Paraná. E-mail: hnapierala@terra.com.br.

procedimentos que permitam-nas tomar decisões adequadas em tempo real e que possibilitem aumentar a satisfação dos clientes, firmando relações duradouras com os mesmos e, ainda, continuarem competitivas e rentáveis. As exigências impostas às empresas são grandes e, ao mesmo tempo, relacionadas com altos riscos como: uma previsão inadequada pode provocar perdas pela superestimação de estoques, o não cumprimento dos prazos estabelecidos pode levar à perda dos clientes. Para se adequar as exigências do mercado, as empresas se utilizam das mais sofisticadas técnicas de planejamento e programação das operações (*scheduling*) que possibilitem a geração de planos otimizados como uma resposta às altas flutuações da demanda e da oferta.

O objetivo deste trabalho é apresentar – de forma resumida – o problema de programação das operações[2] (*scheduling*) – sem intenção de analisar os métodos de solução propostos pela metodologia clássica para cada tipo de problema de *scheduling* – e, a partir destas considerações, apresentar os procedimentos para o *scheduling* de *flow shop permutation* através do algoritmo genético que inclui a busca local.

2 REFERENCIAL TEÓRICO

Em uma economia globalizada, a necessidade de técnicas eficientes de planejamento e controle da produção, tanto em um ambiente industrial quanto em relação aos serviços, é cada vez mais imprescindível. O objetivo geral de planejamento e controle da produção em uma indústria, por exemplo, é gerar um comportamento no qual o fluxo de materiais seja coordenado e permaneça no sistema produtivo por menor tempo possível. Aliás, o planejamento e controle da produção constitui uma variável instrumental que influencia em todas as prioridades competitivas, como: prazo de entrega, qualidade, flexibilidade e custo**s** (MUHLEMANN *et al.*, 2001). Entretanto, na prática, o planejamento e controle da produção caracteriza-se por um frequente atraso no cumprimento dos prazos e em um volume excessivo de trabalho.

---

[2] Os termos *scheduling* e programação das operações serão usados neste estudo como sinônimos. O termo programação, isolado, tem hoje uma conotação ligada à programação de computadores com linguagens de programação; também os termos *schedule* e programa serão usados como sinônimos, referindo-se ao produto da atividade *scheduling*, programação das operações.

Conforme Muhlemann *et al.* (2001, p. 379) o planejamento e controle da produção deve levar em consideração duas restrições importantes e antagônicas: tempo e capacidade produtiva de um recurso. O uso criterioso de recursos e do tempo – a programação das operações como ponto central da economia – tem influenciado o desenvolvimento dos procedimentos computadorizados objetivando a otimização dos recursos e do tempo, não só ao nível de programação da produção de curto prazo, como também para corrigir as oscilações que acontecem entre a demanda e oferta.

Apesar dos avanços tecnológicos, o *scheduling* continua a ser um problema complexo, que se apoia em um modelo genérico, pois as suas características e particularidades variam de caso para caso, devido às diferenças das instalações, processos, produtos e ao ambiente do negócio.

2.1 CONCEITO

Na bibliografia básica encontram se os mais diversos conceitos de programação das operações. Davis *et al.* (2001, p.540) define a programação das operações como: “uma distribuição temporal utilizada para distribuir atividades, utilizando recursos ou alocando instalações”. O objetivo da programação das operações é desagregar a programação mestra de produção em atividades semanas, diárias ou horárias ou, em outras palavras, especificar a carga de trabalho do sistema produtivo planejada para um lapso de tempo muito curto.

O problema de *scheduling*, em geral, é apresentado da seguinte forma (SYSLO *et al*, 1999; BLAZEWICZ *et al*, 1994):

- há um conjunto de n processos, ou tarefas $J = \{J_1, J_2, ..., J_n\}$;
- cada tarefa J_j consiste na execução de k_j operações $O_j = \{O_{1j}, O_{2j}, ..., O_{kj}\}$;
- as operações são processadas por, um conjunto de m máquinas $P = \{P_1, P_2, ..., P_m\}$.

As tarefas com as suas operações caracterizam se pelos seguintes parâmetros:

- cada tarefa tem uma data de chegada, r_j ou data para o processamento e uma data limite de entrega, d_j, sendo $r_j \leq d_j$;
- a cada tarefa pode ser associado um *peso*, w_j, por exemplo, um custo por unidade de tempo;
- pode ser especificada a precedência entre as operações de cada tarefa: assim $O_{aj} < O_{bj}$ indica que O_{bj} só poderá ter início após a conclusão de O_{aj};
- há estoques intermediários, ou *buffers*, entre os processadores em cada tarefa. Ao assumir a capacidade ilimitada, uma tarefa, após o término em um processador, pode esperar até que se inicie o processamento seguinte. Pressupondo, porém, a capacidade zero de *buffers,* ou *no-wait*, a atarefa não podem esperar entre dois processadores consecutivos;
- para cada operação, O_{ij} há um tempo de processamento, p_{ij};
- para cada operação O_{ij} associa-se um tempo de início s_{ij} cujo valor numérico será estabelecido no *scheduling*. O domínio de valores possíveis para cada s_{ij} pode ser definido através de uma data inicial que começa mais cedo r_{ij} e uma data inicial que começa mais tarde rt_{ij}. Estas datas podem ser derivadas da data de início r_j, com $r_{1j} = r_j$, e da data limite d_j, com $rt_{kj} + p_{kj} = d_j$ que não deve extrapolar, sob pena de o processo J_j sofrer atraso. Também pode-se definir a data limite, d_{ij} para cada operação O_{ij}. Os parâmetros s_{ij} só terão valor quando o programa das operações tiver sido construído.

Alguns parâmetros úteis para traduzir critérios de *scheduling* são derivados dos anteriores. Destacam-se para cada processo J_j os seguintes parâmetros:

- tempo de fim, ou *completion time*, C_j;
- tempo de espera, ou *waiting time*, $w_j = C_j - (r_j + \sum p_{ij})$;
- tempo de fluxo, ou *flow time*, soma dos tempos de espera e processamento $F_j = C_j - r_j$);
- atraso relativo, ou *lateness*, $L_j = C_j - d_j$;
- atraso absoluto, ou *tardiness*, $T_j = \max\{C_j - d_j, 0\}$;

- tempo de recurso parado, ou *idle time*, $I_q = C_{\max} - \sum p_{iq}$.

O tempo de fluxo da tarefa é a medida do tempo da tarefa no sistema entre a sua chegada e a sua saída do sistema. O atraso relativo reflete a relação entre o tempo de término e a data de entrega e pode ser negativo, se a tarefa é foi terminada mais cedo. O atraso negativo é mais desejável do que o positivo. Se o custo de processamento de todas as tarefas depende exclusivamente do atraso positivo, e não tem ganho do término da tarefa mais cedo, então, o atraso pode ser tratado como a medida de funcionamento do sistema.

Os problemas de programação das operações são, geralmente, classificados em função de três ambientes de produção:

a) Ambiente aberto (*open shop*). Se as operações de uma tarefa não são restritas a uma sequencia definida, então, o problema é chamado de problema de sequenciamento em ambiente aberto. No sistema aberto de produção a ordem de processamento das operações não é especificada ou preestabelecida para o processamento;

b) Ambiente de fluxo de operações (*flow shop*). O número de operações por tarefa, ou processo k_j, é igual ao número de m processadores e as operações usam os processadores (máquinas) sempre na mesma ordem. Quer dizer, para cada processo J_j, é executada primeiro a operação O_{1j} usando o processador P_1, depois a operação O_{2j} usando o processador P_2, por último a operação O_{mj} usando o processador P_m. Este é o tipo de produção característico das linhas de montagem nas quais os trabalhadores e os postos de trabalho representam os processadores ou máquinas;

c) Ambiente intermitente (*job shop*). O número e a ordem de execução das operações podem variar de tarefa para tarefa (embora seja preestabelecida). É o tipo de produção característica da produção por lotes e da produção unitária.

2.2 CRITÉRIOS DE *SCHEDULING*

O objetivo principal de um problema de *scheduling* é a otimização de uma função objetivo, também denominada critério de seleção, ou medida de qualidade,

que pode variar de acordo com as necessidades do sistema de produção. Segundo Graves (1981) os critérios para a avaliação de programas de operações podem basear-se em uma das medidas de custo e/ou de eficiência. No entanto, na maior parte da bibliografia encontram se trabalhos com um único critério de avaliação.

Rinnooy Kan (1976) e Syslo *et al.* (1999) descrevem alguns dos critérios de avaliação, dentre os quais destaca-se a minimização do tempo máximo do término. Esta medida é também chamada de comprimento da sequência, ou *makespan*, $C_{\max}$. É o tempo de fim programado para a última tarefa, contado a partir do início de programação das operações: $C_{\max} = \max\{C_j\}$. Segundo Blazewicz (1994) a minimização de $C_{\max}$ está associada à utilização eficiente dos recursos produtivos enquanto a minimização da soma dos tempos de fluxo das tarefas (*total flow time*) está associada à redução do estoque em processamento.

Segundo Syslo *et al.*(1999) o *makespan*, C_{max} é a medida especificamente relacionada com o critério mais generalizado, denominado custo máximo,

$$f\_max = max\{f\_j\ (C\_j\ ):\ \ j = 1,2, \dots, n\}.$$

2.3 MÉTODOS DE SOLUÇÃO

Até agora foi considerado o problema de *scheduling* na sua forma ampla, generalizada, mas, a partir deste momento, o foco será deslocado para o problema de *scheduling* do *flow shop permutation*, que é o caso específico do problema de *scheduling* de um sistema de produção em que a sequência tecnológica das n máquinas é a mesma para todas as m tarefas e a ordem em que cada máquina processa as tarefas também será a mesma para todas as máquinas. O problema consiste em obter uma sequencia de tarefas que otimize uma medida de desempenho. Neste trabalho optou-se para minimizar o tempo total de programação C_{max}, o makespan que é associado à utilização eficiente dos recursos produtivos.

Na bibliografia encontram-se as mais diversas técnicas para solucionar este problema, dentre quais destacam se a programação matemática, como a programação linear inteira proposta por Selen e Hott (1986) e *branch and bound* de Ignalle Schrage (1965). No entanto, o problema de *scheduling* – classificado como *NP-hard* – somente pode ser resolvido de uma forma otimizada para os casos de pequeno porte. Outros métodos utilizados para encontrar a solução, embora não

garantam a solução ótima, são os métodos heurísticos, dentre os quais destacam se os trabalhos de Palmer (1965), Campell *et al*. (1970) e Nawaz *et al*. (1983). Além dos métodos heurísticos construtivos, foram desenvolvidas heurísticas melhorativas, que se destacam pelos procedimentos de busca de vizinhança de maior complexidade, tais como a Busca Tabu, proposta por Moccellin e Nagano (1998), e *Simmulated Annealing* apresentado por Ogbu e Smith (1990) e Osman e Potts (1989). Reeves (1995) apresentou o Algoritmo Genético para solucionar o problema de *scheduling* de *flow shop permutation*. Recentemente, foram propostas metaheurísticas híbridas que, para serem mais eficientes, combinam dois procedimentos na busca de solução. No caso da programação de operações no ambiente de *flow shop* destacam-se os trabalhos de Yamada (2003), Buzzo e Moccelin (2000), que usaram métodos híbridos algoritmo genético – *simulated annealing* no problema de *scheduling flow shop permutation* com o objetivo de minimizar o *makespan*. Uma revisão completa do problema de *scheduling* de *flowshop* com critério de *makespan* foi realizada recentemente por Reza Hejazi e Sagnafian (2005).

3 MÉTODO PROPOSTO

Neste trabalho foi utilizada uma técnica denominada Algoritmo Evolutivo Híbrido (AEH), que combina o algoritmo genético com a busca local, que considerada o procedimento 2-opt.

No desenvolvimento do algoritmo genético, objetivando solucionar o problema da programação das operações no ambiente de *flow shop permutation* $FP||C_{max}$, são considerados os seguintes aspectos:

a) estrutura do cromossomo: cada solução do problema de *scheduling* das tarefas pode ser representada por um cromossomo $\Delta^k = \delta_1^k, \delta_2^k, .., \delta_N^k)$, como vetor do conjunto dos números naturais de dimensão N, – de uma lista de tarefas –, no qual o gene $\Delta^k = [\delta_a^k]$ indica o local da tarefa $a \in N$ na sequencia da solução. Assim, o cromossomo representa uma solução do problema de *scheduling*. A população inicial de cromossomos é gerada aleatoriamente;

b) avaliação do *fitness*: o *fitness* de um cromossomo está vinculado à sua capacidade de adaptação ao meio ambiente. Neste algoritmo, adotou-se a função *fitness* transformando a função objetivo do problema:

$$\Phi(\Delta^k) = \psi - \frac{C_{max}(\Delta^k)*P}{\sum_{r=1}^{P} C_{max}(\Delta^k)}, \quad (1)$$

onde $C_{max}(\Delta^k)$ é o valor da função objetivo do k-ésimo indivíduo; P é o tamanho da população e ψ é um valor muito elevado para assegurar a não negatividade do valor de fitness. O valor ψ, adotado neste trabalho, é o valor máximo do segundo termo da equação (1). Assim, o valor do *fitness* em relação ao valor máximo da função objetivo é igual a zero;

c) processo de seleção natural: em cada iteração – dentre os cromossomos mantidos na população – dois são escolhidos aleatoriamente para se reproduzirem. A cada processo de reprodução dois novos cromossomos são gerados, os quais substituem os piores armazenados em *t*, caso a avaliação do *fitness* assim o indique. Este processo garante que, pouco a pouco, a população seja melhorada. A escolha de um cromossomo para a reprodução é feita na base da probabilidade de reprodução, expressa da seguinte forma:

$$p_r(\Delta^k) = \frac{\Phi(\Delta^k)}{\sum_r \Phi(\Delta^k)}, \quad (2)$$

onde Δ^krepresenta o k- ésimo indivíduo pertencente à população P^te $\Phi(\Delta^k)$ o valor de sua adaptação. Uma vez determinada a probabilidade de reprodução $p_r(\Delta^k)$ e a sua distribuição – *Cumulative Distribution Funtion* – $P_r\Delta^k$, faz-se a escolha do cromossomo para a reprodução, a partir da seguinte expressão:

$$rnd(\,) \leq P_r(\Delta^k) = \sum_{i=1}^{k} p_r(\Delta^i), \quad (3)$$

onde $rnd(\,)$ é um número pseudo-aleatório, uniformemente distribuído no intervalo [0,1];

d) processo de reprodução: o cruzamento entre cromossomos é realizado pela operação de *crossover*. Neste procedimento, um índice *q* – que indica a posição de quebra da cadeia dos cromossomos geradores – é escolhido aleatoriamente e, através do operador de cruzamento de mapeamento parcial, PMX – *partially-mapped crossover*, é realizada uma re-combinação das partes, de modo que uma sequência parcial de um cromossomo é copiada e, na medida do possível, é mantida a sequência e a posição de operações das tarefas de um outro cromossomo:

$$crossover_1(\Delta^k, \Delta^l) = \left(\delta_1^k, \delta_1^k, .., \delta_q^k, \delta_{q+1}^l, \delta_{q+2}^l, .., \delta_n^l\right)$$

e

$$crossover_2(\Delta^k, \Delta^l) = \left(\delta_1^l, \delta_1^l, .., \delta_q^l, \delta_{q+1}^k, \delta_{q+2}^k, .., \delta_n^k\right)$$

e) processo de mutação: o operador de mutação, por meio de uma troca de dois genes de um cromossomo, é usado para diversificar a população e escapar da solução ótima local. Esta operação – *reciprocal exchange* – é realizada através de uma escolha aleatória de dois genes que modificam a estrutura cromossomo Δ^k, com a probabilidade p_m.

Procedimentos baseados no cruzamento exigem muitos cálculos e os procedimentos baseados somente na mutação – *insertion* ou *recíprocal exchange* – não garantem o modo efetivo para evitar os ótimos locais sem adicionais mecanismos probabilísticos de seleção (MICHALEWICZ e FOGEL, 2006). Estas dificuldades desencadearam o processo objetivando desenvolver os operadores de procura local adicionais aos esquemas evolutivos. A diferença entre o algoritmo evolutivo básico e o algoritmo enriquecido pelo otimizador local é pequena, porém significativa. Cada indivíduo de uma população inicial, ou descendente desta, é submetido à avaliação local otimizada. Deste modo, os ótimos locais substituem as soluções originais. O espaço de busca do algoritmo evolutivo é limitado somente às soluções locais otimizadas de uma função de avaliação e de heurística de busca local.

Existem diversas maneiras para se elaborar os algoritmos evolutivos que contém operadores de busca local. Neste trabalho é considerado o procedimento 2-opt para substituir cada solução – sequencia de tarefas – de uma população – inicial e descendente – por uma solução ótima local. Este procedimento consiste na troca de duas tarefas não vizinhas entre si. Se a nova sequência é melhor do que a original, então, é realizada a substituição, em caso contrário, a sequência original é mantida e denominada 2-*optimal* e o procedimento termina.

O algoritmo proposto, que opera no espaço de soluções alternativas e tem como base a evolução, segue os seguintes passos:

Passo 1 – *Inicialização*: gere uma lista $\Delta = (\Delta^1, \Delta^2, .., \Delta^N)$, com N cromossomos – cada um representando uma sequência de tarefas a serem processadas e calcule o *fitness* $\Phi(\Delta^k)$, $\forall \Delta^k \in \Delta$; faça o índice de interações $t = 0$ e defina o número máximo de interações τ;

Passo 2 – *Otimização local*: realize o 2-opt para a população inicial;

Passo 3 – *Critério de parada*: se $t = \tau$, então pare e apresente o melhor cromossomo;

Passo 4 – *Seleção*: selecione dois cromossomos, de acordo com a probabilidade de reprodução expressa na equação (2);

Passo 5 – *Crossover*: realize o $crossover_1(\Delta^k, \Delta^l)$ e $crossover_2(\Delta^k, \Delta^l)$;

Passo 6 – *Mutação*: efetue a mutação através do operador *reciprocal exchange*, com a probabilidade p_m;

Passo 7 – *Otimização local*: realize o 2-opt para a população descendente;

Passo 8 – *Avaliação*: calcule o *fitness* dos cromossomos descendentes, $\Phi(\Delta^k)$, $\forall \Delta^k \in \Delta$; insira-os em P^t e elimine os cromossomos de menor valor de adaptação; faça $t = t + 1$ e volte ao passo 2.

4 EXPERIÊNCIA COMPUTACIONAL

O método proposto, denominado AEH – Algoritmo Evolutivo Híbrido, foi implementado sob a forma de um sistema computacional, em linguagem DELPHI 7.0 para Windows, e rodado em um microcomputador Turion 64 x2 de 1.60GHz, com 1GB de memória RAM. O desempenho do modelo foi testado em duas situações: em uma análise comparativa com o problema apresentado por Taillard (1993) e em uma análise comparativa com dois algoritmos: *Simulated Annaeling* e Algoritmo Genético. Para que a análise fosse conduzida de maneira mais adequada, os dois últimos algoritmos também foram descritos na linguagem de programação do Object Pascal/Delphi.

Antes da aplicação do modelo foi necessário estimar os parâmetros do algoritmo genético. Os parâmetros para determinar a performance do algoritmo são: o tamanho da população, a probabilidade de *crossover*, a probabilidade de mutação, a probabilidade de inserção, a probabilidade de eliminação e o critério de parada. Nas simulações foram utilizados os seguintes parâmetros: o tamanho da população $P^t = 500$ indivíduos para AEH e $P^t = 5000$ indivíduos para o AG; a probabilidade de *crossover* $p_c = 0{,}7$; a probabilidade de mutação $p_m = 0{,}04$ e a probabilidade de inserção e de eliminação $p_i = p_d = 0{,}01$.[3] O critério de parada do processo evolutivo baseia-se no número máximo de interações $t = 100$, isto é, as interações são interrompidas quando o número de interações chega a 100 para o AEH e $t = 500$ para AG.

[3] Os parâmetros utilizados neste trabalho foram sugeridos por ARABAS (2001:277), que, a partir de estudos realizados em diversas implementações, propôs a utilização dos mesmos.

Para o algoritmo *Simmulated Annealing* foram estabelecidos os seguintes parâmetros de controle: a temperatura inicial $t_o = 20$ e o parâmetro da função de resfriamento $r = 0{,}90$.

Os testes foram agrupados em classes, de acordo com o número de tarefas $n \in \{20{,}50{,}100\}$ e o número de máquinas $m \in \{5{,}10{,}20\}$. Todos os testes foram gerados aleatoriamente, com o tempo de processamento p_{ij}, da tarefa i, na máquina j e o número inteiro distribuído uniformemente no intervalo de [1, 99], conforme o problema apresentado por Taillard (1993).

Como nenhum dos algoritmos em análise pode garantir a solução ótima do problema, os resultados foram analisados em termos relativos quanto ao desvio e à qualidade. O primeiro indicador mede o desvio relativo médio, DR_h para cada algoritmo h, que é obtido da seguinte maneira:

$$DR_h = \frac{(C_{max} - C^*_{max})}{C^*_{max}} \quad (4)$$

onde, C_{max} é o *makespan* obtido de uma determinada categoria do teste; C^*_{max} é o melhor *makespan* obtido para aquela categoria.

O segundo indicador, a qualidade relativa QR é obtido pela expressão:

$$QR = \frac{C_{max}}{C^*_{max}} \quad (5)$$

onde, C_{max} é o resultado para uma dada categoria de teste e C^*_{max} é o melhor *makespan* obtido entre os resultados de uma categoria.

Os resultados das simulações estão demonstrados na Tabela 1,2 e 3.

Tabela 1: Resultado do desempenho em termos de qualidade relativa

Tarefas (n)	Máquinas (m)	Qualidade Relativa Média AEH	A.G.	S.Ann
20	5	1.0000	1.0000	1.0046
	10	1.0000	1.0427	1.0466
	20	1.0000	1.0444	1.0699
50	5	1.0000	1.0122	1.0322
	10	1.0000	1.0591	1.0825
	20	1.0000	1.0651	1.0935
100	5	1.0000	1.0060	1.0265
	10	1.0000	1.0664	1.0794
	20	1.0000	1.0747	1.0953

Dos resultados observados da Tabela 1 e 2 pode-se dizer que o método heurístico hibrido AEH tem um melhor desempenho do que algoritmo genético geral

e o *simulated annealing*. Isto demonstra que as modificações realizadas nos procedimentos do algoritmo genético – otimização local – melhoram substancialmente o algoritmo. Esta melhoria varia de 0 a 7,47% e, em comparação à *simulated annealing*, de 0,46 a 9,54%, conforme a Tabela 1.

Tabela 2: Resultado da análise em termos de desvio relativo médio

Tarefas (n)	Máquinas (m)	Desvio Relativo Médio (%) AEH	A.G.	S.Ann
20	5	0.00	0.07	1.14
	10	0.09	0.64	2.37
	20	0.29	1.23	1.25
50	5	0.00	0.85	0.65
	10	0.12	0.45	0.84
	20	0.19	0.73	0.83
100	5	0.00	0.69	0.45
	10	0.38	0.26	0.76
	20	0.30	0.77	0.49

A qualidade da solução do AEH não é tão afetada por problemas estruturais, nem pela dimensão do problema, como o algoritmo genético geral e o *simulated annealing* (Tabela 2). Isto indica que o procedimento AEH é mais robusto.

Tabela 3: Resultado da análise comparativa com o problema de Taillard

Resultados do problema de Taillard									
	20x20			50x20			100x20		
Número	AEH	UB	LB	AEH	UB	LB	AEH	UB	LB
1	**2228**	2297	1911	**3797**	3886	3480	6388	6345	5851
2	**2094**	2100	1711	3739	3733	3424	6410	6323	6099
3	**2320**	2326	1844	3823	3689	3351	6579	6385	6099
4	2287	2223	1810	3760	3755	3336	6352	6331	6072
5	**2238**	2291	1899	3723	3655	3313	6563	6405	6009
6	**2218**	2226	1875	3758	3719	3460	6558	6487	6144
7	**2192**	2273	1875	3750	3730	3427	6555	6393	5991
8	**2133**	2200	1880	**3742**	3744	3883	6561	6514	6084
9	**2096**	2237	1840	**3746**	3790	3457	6417	6386	5979
10	2189	2178	1900	**3761**	3791	3438	**6535**	6544	6298

AEH: o melhor *makespan* obtido pelo algoritmo evolutivo híbrido
UB: o melhor valor de *makespan* conhecido e referenciado em OR-Library
LB: limite inferior teórico

A tabela 3 resume as estatísticas de desempenho do algoritmo evolutivo híbrido AEH para um subconjunto dos problemas proposto por Taillard, juntamente

com os limites inferior e superior, referenciado em OR-Library[4]. Limites superiores são os *makespans* mais conhecidos atualmente; a maioria deles foi encontrada através do procedimento *branch and bound,* com tempo computacional desconhecido. Ao todo, 30 execuções foram concluídas para cada problema nas mesmas condições, mas com sementes de números aleatório diferentes. Cada execução leva entre 1 minuto e 20 segundos; 3 minutos e 22 segundos; 8 minutos e 18 segundos de tempo de CPU, respectivamente, para cada categoria: 20x20, 50x20 e 100x20 do problema.

A qualidade da solução para problemas de 20x20 são notáveis: os resultados obtidos pelo método heurístico híbrido AEH tem um desempenho aceitável, se comparados com o conjunto de dados para a maioria dos problemas, e alguns resultados (destacados em negrito) são ainda melhores do que os melhores resultados relatados atualmente na biblioteca de pesquisa operacional (OR-Library). Os resultados para problemas maiores não são tão expressivos como aqueles para problemas de tamanho 20x20, mas estes são, ainda, suficientemente bons para suportar o algoritmo evolutivo híbrido AEH. A deterioração da qualidade de solução é, provavelmente, devido à crescente complexidade do cálculo da busca local.

5 CONCLUSÕES

As empresas sempre estão interessadas em melhorar a utilização de seus recursos ou atender melhor seus clientes. Cada cliente exige que o seu pedido seja tratado como prioritário. Exige, ainda, que o seu pedido seja realizado o mais rápido possível, mas também que seja mantido o padrão de qualidade desejável. O não cumprimento destas expectativas pode significar a perda do cliente em favor dos concorrentes. Esta pressão do tempo, do custo e da qualidade, associada à diversificação da produção, exige da empresa a otimização dos planos operacionais. Os planos operacionais inadequados podem aumentar os custos da produção ao ponto de superar os preços de venda; o não cumprimento dos prazos de entregada, por sua vez, pode causar as penalidades de atraso ou de adiantamento, o que, *per saldo,* torna a produção dispendiosa.

[4] OR-Library é uma coleção de um conjunto de dados de testes para diversos problemas de Pesquisa Operacional. OR-Library foi escrita originalmente por BEASLEY, J.E. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, v. 41(11) p.1069-1072, 1990.

O propósito deste trabalho foi apresentar resumidamente o problema de programação das operações e, a partir destas considerações, apresentar os procedimentos para o *scheduling* de *flow shop permutation.* Foram investigadas várias abordagens para resolver problemas de programação das operações no ambiente de *flow shop*. O estudo incluiu as metaheurísticas – algoritmo genético e *simulated annealing* – que serviam como ponto de referência para a elaboração do método proposto. Os métodos heurísticos descritos apresentam vantagens concretas, visto que podem ser usados em combinação com outros métodos, bem como por serem flexíveis, ou seja, por serem capazes de tratar, de uma forma mais eficiente, a função objetivo e as restrições do problema. Uma característica comum destes algoritmos é a de eles tendem a convergir para uma solução que é o ótimo global no espaço do problema, embora não garantam uma solução ótima. Esta propriedade de convergência parece ser importante, na medida em que permite a melhoria, de uma forma iterativa das soluções sub-ótimas. Em termos de qualidade, é difícil dizer qual das heurísticas é a melhor, pois própria parametrização dos procedimentos faz com que seja praticamente impossível se chegar a esta definição.

As principais características do método proposto incluem o uso efetivo do conceito da evolução expressa através do algoritmo genético, bem como o conceito da busca local. A utilização destes conhecimentos específicos do problema melhora o desempenho do método proposto, pois o conhecimento específico do problema permite reduzir a dimensão do espaço de pesquisa e se limita à convergência da solução a partir dos ótimos locais. As melhores soluções são obtidas por meio de testes extensivos, incluindo a análise das propriedades do problema e a parametrização adequada.

Atualmente, a automação de fábrica é tão avançada que cada máquina tem sensores de autodiagnóstico (*Production Data Acquisition*), bem como uma interface com a rede de computadores, LAN, e monitoramento remoto *on-line*. A flexibilidade e a precisão de controle de máquinas para a reconfiguração e reprogramação também são melhoradas. Por conseguinte, justifica se a necessidade de gerar um procedimento eficiente de programação que encontre a solução em tempo razoável. No entanto, infelizmente, o problema de *scheduling* no ambiente de *flow shop* investigado neste trabalho, pode ser demasiadamente simplista em comparação aos problemas reais da produção, com uma ampla variedade de restrições, funções objetivo mais flexíveis e com as características mais dinâmicas. Espera-se estender

a abordagem apresentada neste trabalho para incorporar configurações mais realistas e acredita-se, ainda, que a maioria das ideias apresentadas neste trabalho possam ser úteis em futuras pesquisas vinculadas a *scheduling*.

REFERÊNCIAS

ARABAS, J. **Wyklady z algorytmow evolucyjnych.** Warszawa: WNT, 2001.

BEASLEY, J. E. OR-Library: distributing test problems by electronic mail. **European Journal of Operational Research**, v. 41, p. 1069-1072, 1990.

BLAZEWICZ, J.; ECKER, K.H.; SCHMIDT, G.; WEGLARZ, J. **Scheduling in Computer and Manufacturing Systems**. Berlin: Springer Verlag, 1994.

BUZZO, W. R.; MOCCELLIN, J. V. Programação da Produção em Sistemas Flow Shop utilizando um Método Heurístico Híbrido Algoritmo Genético Simulated Annealing. **Gestão & Produção**, v.7, n.3, p.364-377, 2000.

CAMPELL, H.G.; DUDEK, R.A.; SMITH, M.L. A Heuristic Algorithm for the n-Job, m-Machine Sequencing Problem. **Management Science**, v.16/B, p.630-637. 1970.

DAVIS, M.M.; AQUILANO, N.J.; CHASE, R.B. **Fundamentos da Administração da Produção**. Trad. Eduardo D'Agord Schan, E. *et al.* 3.ed. Porto Alegre: Bookman Editora, 2001.

GRAVES, S.C. A Review of Production Scheduling, **Operation Research**. v. 28, p. 646-675, 1981.

IGNALL, E.; SCHRAGE, L.E. Aplication of Branch and Bound Technique to some Flow-Shop Problem. **Operation Research**, v.13, p. 400-412, 1965.

MICHALEWICZ, Z.; FOGEL, D. **Jak to rozwiazac czyli nowoczesna heurystyka**. (trad.) Schubert. Warszawa: Wydawnictwo Naukowo-Techniczne, 2006.

MOCCELLIN, J.V.; NAGANO, M.S. Evaluating the Performance of Tabu Search Procedures for Flow Shop Sequencing. **Journal of the Operation Research Society**, v.49, p. 1296-1302, 1998.

MUHLEMANN, A.P.; OAKLAN, J.S.; LOCKYER, K.G. **Zarzadzanie: produkcja i uslugi**. Red.trad. Jaroslaw Soltys, 3 ed. Warszawa: Wydawnictwo Naukowe PWN, 2001.

NAWAZ, M.; ENSCORE, E.E.; HAM, I. A Heuristic Algorithm for m-Machine, n-Job Flow-Shop Sequencing Problem. **OMEGA**, v.11, p. 91-95, 1983.

OGBU F. A.; SMITH, D. K. The application of the simulated annealing algorithm to the solution of the *n/m*/*Cmax* flowshop problem. **Computers Opns Res.***,* v. 17 (3) p. 243-253, 1990.

OSMAN, I.H., POTTS, C.N. (1989) Simulated annealing for permutation flowshop problem. **Omega**, v.17 (6), p. 551-557, 1989.

PALMER, D.S. Sequencing Job through a Multi-stage Process in the Minimum Total Time – A Quick Method of obtaining a Near Optimum. **Operation Research Quarterly**, v.16, p.101-107, 1965.

REEVES, C.R. A Genetic Algorithm for Flowshop Sequencing. **Computers & Operation Research**, v.22, p.5-13, 1995.

REZA HEJAZI, S.; SAGNAFIAN, S. Flowshop-scheduling problems with makespan criterion: a review. **International Journal of Production Research***,* v. 43, (14), p. 2895–2929 (2005).

RINNOOY KAN, A.G.H. **Machine Scheduling Problems, Classification, Complexity and Computations**. Hague: Nijhoff, 1976.

SELEN, W.J.; HOTT, D.D. A Mixed-Integer Goal Programming Formulation of the Standard Flow-Shop Scheduling Problem. **Journal of the Operational Research Society**, v.37, p. 1121-1128, 1986.

SYSLO, M.M.; DEO, N.; KOWALIK, J.S. **Algorytmy optymizacji dyskretnej**. Trad. M. M. Syslo *et al*, 3 ed. Warszawa: Wydawnictwo Naukowe PWN, 1999.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, v.64, p. 278-285, 1993.

YAMADA, T. **Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problem**. Doctor Thesis, Kyoto University, Kyoto, Japan, 2003.

Artigo recebido em: Março/2014
Aceito em: Junho/2014