ALGORITMO GENÉTICO BI-OBJETIVO APLICADO AO PROBLEMA DE ESCALONAMENTO DE TAREFAS FLOW SHOP

Miguel Angel Fernández Pérez

miguelfp177@yahoo.com.br

Fernanda Maria Pereira Raupp

fraupp@puc-rio.br

Departamento de Engenharia Industrial / PUC-RIO
Rua Marquês de São Vicente, 225, Gávea - Rio de Janeiro, RJ - Brasil - 22451-900

RESUMO

Neste trabalho, realizamos um estudo preliminar da aplicação de um algoritmo genético básico para resolver o problema de escalonamento *flow shop*, considerando a minimização simultânea do *makespan* e do tempo de fluxo médio. Testamos um algoritmo genético bi-objetivo, usando três distintas funções aptidão, em duas instâncias de problema. Analisamos a sensibilidade do algoritmo em gerar resultados, que são soluções aproximadamente não-dominadas ou eficientes, em relação à escolha da função aptidão em termos da quantidade de pontos gerados e de sua dispersão sobre a fronteira de Pareto.

PALAVARAS CHAVE. Escalonamento flow shop, Algoritmo genético multiobjetivo, Não-dominância.

ABSTRACT

In this work, a preliminary study of applying a basic genetic algorithm to solve the flow shop scheduling problem is conducted, considering the simultaneous minimization of the makespan and the mean flow time. We tested a bi-objective genetic algorithm, using three different fitness functions, in two instances the problem. We analyze the sensitivity of the algorithm in generating results, which are approximately non-dominated or efficient solutions, in relation to the choice of the fitness function in terms of the amount of generated points and their dispersion on the Pareto frontier.

KEYWORDS. Flow shop scheduling, Multiobjective genetic algorithm, Non-dominance.

1. Introdução

O escalonamento *flow shop* é um sistema de trabalho de n tarefas em m máquinas em série, onde cada tarefa tem que ser processada em cada uma das m máquinas. Todas as tarefas devem seguir o mesmo roteiro, ou seja, elas têm que ser processadas, primeiro na máquina 1, depois na máquina 2, e assim por diante. Após a conclusão de uma tarefa em uma máquina, a tarefa se junta à fila da próxima máquina. O problema de escalonamento *flow shop* é classificado como NP-difícil para a maioria dos problemas clássicos. Já são considerados NP-difíceis os problemas seguintes: $F_2 \,||\, \sum C_j$, $F_2 \,||\, L_{\max}$ e $F_3 \,||\, C_{\max}$ (PINEDO, 2008).

Muitos esforços têm sido feitos no sentido de resolver o problema de escalonamento *flow shop* envolvendo diferentes metodologias. Por exemplo, Widmer e Hertz (1989) propuseram um método heurístico para resolver o problema de escalonamento *flow shop* com o objetivo de minimizar apenas o *makespan*. Este método é composto de duas fases: a primeira fase considera uma sequência inicial correspondendo a uma solução do problema do caixeiro viajante, e a segunda tenta melhorar essa solução usando técnicas de busca tabu. Já o trabalho de Ho (1995) apresenta uma heurística para minimizar apenas o tempo de fluxo médio para o problema de escalonamento *flow shop*. Um estudo de simulação é realizado para testar a eficácia da heurística proposta, comparando-a com outros métodos.

Considerando, agora, mais de um objetivo a ser otimizado no problema de escalonamento *flow shop*, destacamos o trabalho de Ponnambalam *et al*. (2004). Seus autores propõem um algoritmo de busca evolutiva multiobjetivo, utilizando um algoritmo que resolve o problema do caixeiro viajante e um algoritmo genético. O algoritmo utiliza uma soma ponderada dos vários objetivos como função aptidão. Os pesos são gerados aleatoriamente a cada geração para permitir uma busca multi-direcional. As medidas de desempenho consideradas incluem minimizar o *makespan*, o tempo de fluxo médio e o tempo de máquina parada. Pasupathy *et al*. (2006) propuseram um algoritmo genético multiobjetivo para encontrar soluções eficientes ou não-dominadas do problema de escalonamento *flow shop*, mediante técnicas de busca local, e considerando os objetivos de minimizar o *makespan* e o tempo de fluxo total. Esse algoritmo faz uso do princípio da não-dominância juntamente com uma métrica de aglomeração, visando superar a dificuldade do algoritmo em gerar uma fronteira eficiente.

Neste trabalho, resolvemos aproximadamente o problema de escalonamento *flow shop* aplicando um algoritmo genético bi-objetivo básico com operadores de seleção, cruzamento e mutação com estratégia de elitismo. Os objetivos considerados são: minimizar o *makespan* para reduzir o tempo de conclusão das tarefas e minimizar o tempo de fluxo médio para reduzir o número médio de tarefas em espera na fila, visto que esses objetivos são classicamente usados em problemas de escalonamento *flow shop* com apenas um objetivo e apresentam conflito por natureza. Consideram-se diferentes possibilidades de definição da função aptidão, em distintos cenários. A sensibilidade do algoritmo é analisada ao gerar resultados, que são soluções aproximadamente não-dominadas ou eficientes, em relação à escolha da função aptidão e à atribuição de seus parâmetros em termos da quantidade de pontos e de sua dispersão sobre a fronteira de Pareto.

2. Otimização Multiobjetivo

O problema de otimização multiobjetivo com r objetivos pode ser definido da seguinte forma: dado um vetor de variáveis de decisão com dimensão n, $x = \{x_1,...,x_n\}$ no espaço de busca X, queremos encontrar um vetor $x^* \in X$ que minimiza simultaneamente as r funções objetivo $f(x^*) = \{f_1(x^*),..., f_r(x^*)\}$. Em geral, X é definido por uma série de restrições e limites de especificação para as variáveis de decisão.

Se todas as funções objetivo são de minimização, uma solução viável x diz-se dominar outra solução viável y se e somente se $f_i(x) \leq f_i(y)$ para $i = 1,...,r$ e $f_j(x) < f_j(y)$ para pelo menos uma função objetivo j. Uma solução viável é dita *Pareto-ótima*, ou *não-dominada*

ou *eficiente*, se não for dominada por nenhuma outra solução viável no espaço de busca. Uma solução Pareto-ótima não pode ser melhorada com relação a qualquer objetivo sem que exista piora para pelo menos algum outro objetivo. O conjunto das soluções não dominadas em X é chamado de *conjunto Pareto-ótimo*, e a imagem de um determinado conjunto Pareto-ótimo, no espaço dos valores dos objetivos, é chamada de *fronteira de Pareto* (KONAKA, COIT & SMITH, 2006).

Em muitos problemas da vida real os objetivos são conflitantes entre si. Assim, uma solução pode ser ótima para um objetivo e não ser ótima para os demais. Portanto, uma solução razoável para um problema multiobjetivo é uma solução que não seja dominada por qualquer outra solução. Melhor seria ter como resultado o conjunto Pareto-ótimo.

3. O problema de escalonamento *flow shop*

3.1 Descrição do problema

Um problema de programação de operações em um ambiente *flow shop* é um problema de programação de produção no qual n tarefas devem ser processadas em um conjunto de m máquinas distintas, tendo a mesma ordem de processamento nas máquinas. O problema consiste em determinar uma sequência de tarefas (ou uma *programação permutacional*) dentre $n!$ sequências possíveis, que é mantida para todas as máquinas, de modo a otimizar uma determinada medida de desempenho da programação, associada geralmente ao fator tempo (BUZZO & MOCCELLIN, 2000).

As hipóteses consideradas no problema de escalonamento de *flow shop* são:
1) cada máquina está disponível continuamente, sem interrupções;
2) cada máquina pode processar apenas uma tarefa de cada vez;
3) cada tarefa pode ser processada por uma máquina de cada vez;
4) os tempos de processamento das tarefas nas diversas máquinas são determinados e fixos;
5) as tarefas têm a mesma data de liberação, a partir da qual, qualquer uma pode ser programada e executada;
6) os tempos de preparação das operações nas diversas máquinas são incluídos nos tempos de processamento e independem da sequência de operações em cada máquina e
7) uma vez iniciadas as operações nas diversas máquinas, elas não devem ser interrompidas.

Observe que um problema de escalonamento *flow shop* envolvendo apenas 10 tarefas apresenta 3.628.800 sequências possíveis de programação.

3.2 Formulação do problema

Para formular o problema foram avaliados os critérios do *makespan* e do tempo de fluxo médio. A otimização do *makespan* reduz o tempo de conclusão das tarefas e resulta numa utilização eficiente dos recursos e a otimização do tempo de fluxo médio reduz o número médio de tarefas em espera na fila. (PASUPATHY *et al.*, 2006). Além disso, a otimização do tempo fluxo médio é importante, porque muitas das medidas de custo, como a rotação de produtos acabados dependem do tempo de fluxo (HO, 1995). Minimizar o tempo de fluxo médio requer que a média de conclusão de todas as tarefas seja a menor possível à custa de uma tarefa que demora um tempo maior em concluir, enquanto minimizar o *makespan* requer que nenhuma tarefa demore muito tempo. Assim minimizar o *makespan* implica maximizar o tempo de fluxo médio. Portanto este problema deve ser tratado como um problema de otimização multiobjetivo, já que tem objetivos conflitantes (LIU *et al.*, 2010).

Portanto, atendendo as necessidades da prática, neste trabalho, tem-se como objetivo encontrar um conjunto de sequências de n tarefas a serem executadas em m máquinas, de modo a minimizar simultaneamente o *makespan* e o tempo de fluxo médio.

Dado o tempo de processamento $p(i, j)$ para a tarefa i na máquina j, para $i = 1,...,n$ e $j = 1,...,m$, e dada uma sequência de tarefas a serem realizadas $J_1 - J_2 - ... - J_n$, então podemos calcular o *makespan* ou tempo de total de processamento, denotado por $C_{\max}$, da seguinte forma (REEVES, 1995):

$$C(J_1,1) = p(J_1,1)$$
$$C(J_i,1) = C(J_{i-1},1) + p(J_i,1), i = 2,...,n$$
$$C(J_1, j) = C(J_1, j-1) + p(J_1, j), j = 2,...,m$$
$$C(J_i, j) = \max\{C(J_{i-1}, j), C(J_i, j-1)\} + p(J_i, j), i = 2,...,n; j = 2,...,m$$
$$C_{\max} = C(J_m, m).$$

O tempo de fluxo médio, denotado por $\overline{F}$, é definido como sendo a média do tempo total de processamento das n tarefas, dado por

$$\overline{F} = \frac{1}{n}\sum_{i=1}^{n} C(J_i, m).$$

Desta forma o problema que será resolvido é do tipo $F_m \,||\, C_{\max}, \overline{F}$.

4. Algoritmo genético multiobjetivo

Algoritmo genético é um método de busca local estocástica que resolve um problema de otimização, fazendo analogia à Teoria da Evolução de Darwin. Ele simula o mecanismo de sobrevivência natural (continuidade) do indivíduo (da solução) mais apto (melhor em relação à função aptidão), onde cada cromossomo (solução) é avaliado por uma função aptidão de acordo com o problema que se deseja resolver (KAMIRI *et al*., 2010).

Algoritmos genéticos têm sido aplicados com sucesso para resolver uma grande variedade de problemas de otimização, e em particular, problemas de escalonamento; fato evidenciado nos trabalhos de Buzzo e Moccellin (2000), Ponnambalam *et al*. (2004), Zhang e Gen (2006), Yang e Tang (2009) e Liu *et al*. (2009), para citar apenas alguns.

Proposto por Murata e Ishibuchi (1995), o algoritmo genético aplicado neste trabalho para resolver o problema de escalonamento *flow shop* bi-objetivo é descrito a seguir. Defina N_{pop}, N_{elite} e N_{gen} como números inteiros positivos.

Passo 0: Gere aleatoriamente uma população inicial de soluções candidatas de tamanho N_{pop}.
Passo 1: Calcule os valores das funções objetivo de cada solução. Atualize o conjunto de soluções Pareto-ótimas.
Passo 2: Calcule o valor de aptidão de cada solução candidata e aplique o operador de seleção na população corrente.
Passo 3: Aplique o operador de cruzamento.
Passo 4: Aplique o operador mutação.
Passo 5: Retire ao acaso N_{elite} soluções do conjunto com N_{pop} soluções geradas pelos operadores genéticos, e as substitua com N_{elite} soluções escolhidas ao acaso no conjunto de soluções Pareto-ótimas.
Passo 6: Se o número máximo de gerações N_{gen} não foi atingido, retorne ao Passo 1. Senão, vá para o passo 7.
Passo 7: Os resultados estão no conjunto de soluções Pareto-ótimas.

4.1 Representação de uma solução

A codificação utilizada para representar cada uma das alternativas possíveis para resolver o problema de escalonamento *flow shop* é um vetor de tamanho n, correspondendo ao número de tarefas a serem executadas. Deste modo, a k-ésima posição do vetor representa a k-ésima tarefa a ser processada.

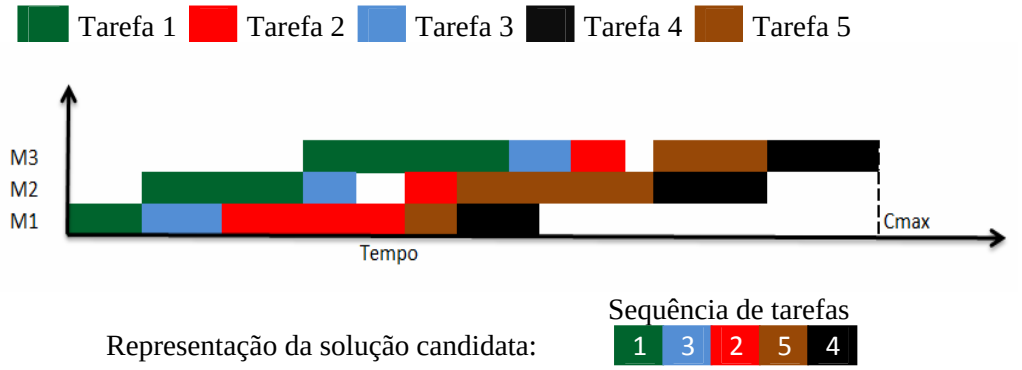

Figura 1. Exemplo de um escalonamento *flow shop* para $n = 5$ e $m = 3$.

Apresentamos na Figura 1 um exemplo da representação de uma solução candidata (cromossomo) na população referente a um problema de escalonamento *flow shop* para $n = 5$ e $m = 3$. Pode-se observar que a tarefa 1 é a primeira a ser inicialmente executada na máquina 1, depois na máquina 2, e por último na máquina 3. No momento que a máquina 1 estiver liberada da tarefa 1, a tarefa 3 é executada, igualmente, no momento que a máquina 2 estiver liberada da tarefa 1, a tarefa 3 será processada, e assim por diante para todas as tarefas. Também, pode-se notar que a tarefa 2 somente começará a ser processada na máquina 2 quando seu processamento na máquina 1 for terminado, o mesmo acontece com a tarefa 5 antes de começar seu processamento na máquina 3.

4.2 Geração da população inicial

A população inicial das soluções candidatas (cromossomos) é gerada de maneira aleatória, onde cada solução é uma permutação das tarefas existentes gerada aleatoriamente.

4.3 Avaliação de soluções

As soluções candidatas (cromossomos) são avaliadas segundo as funções objetivo, as quais são combinadas em uma única função denominada função aptidão. O valor da função aptidão na solução candidata indica a chance que ela tem em gerar novas soluções. Quanto melhor a aptidão, maior é a chance.

Ponnambalam *et al.* (2004) propuseram um algoritmo genético multiobjetivo para resolver o problema de escalonamento *flow shop*, onde a função aptidão em cada solução candidata x é calculada como $a_1(x) = 1/(1 + w_1 f_1(x) + w_2 f_2(x))$, onde w_1 e w_2 são números aleatórios não negativos, tal que $w_1 + w_2 = 1$, e f_1 e f_2 são as funções objetivo.

Yang e Tang (2009) propuseram um algoritmo genético multiobjetivo para resolver o problema de escalonamento típico de um Sistema de Planejamento e Programação Avançado (APS). Nesse trabalho, a aptidão de cada solução x é calculada como $a_2(x) = w_1 / f_1(x) + w_2 / f_2(x)$, onde w_1 e w_2 são constantes e iguais a 0,5 e f_1 e f_2 são as funções objetivo.

Zhang e Gen (2006) propuseram um algoritmo genético multiobjetivo para resolver o problema de seleção de recursos e escalonamento de operações similar ao modelo multiobjetivo de um sistema APS. Nesse algoritmo, a aptidão de cada solução x é calculada como $a_3(x) = w_1 f_1(x) + w_2 f_2(x)$, onde w_1 e w_2 são números não negativos, tal que $w_1, w_2 \leq 1$ e f_1 e f_2 são as funções objetivo. Note que a função aptidão é uma soma ponderada das funções objetivo.

Mais adiante, na seção 5, será feita uma análise de distintos cenários para conhecer o desempenho do algoritmo genético em relação à escolha da função aptidão e dos valores w_1 e w_2.

4.4 Seleção

A seleção é o processo de escolha das melhores soluções candidatas (cromossomos mais aptos) da população. As soluções candidatas são comparadas em termos de seus valores da função aptidão. A seleção por torneio é usado neste trabalho. O operador de seleção por torneio escolhe aleatoriamente 2 soluções candidatas e as compara, escolhendo a solução com maior valor de aptidão (PONNAMBALAM *et al.*, 2009).

4.5 Cruzamento

O Cruzamento é o operador de troca entre um par de soluções candidatas (cromossomos) da geração corrente, que por sua vez gera um par de soluções candidatas descendentes semelhante às soluções candidatas pais. A aplicação do operador cruzamento ocorre após a seleção dos indivíduos da população corrente. A propagação de características dos mais aptos entre soluções candidatas geradas a cada nova geração é a idéia desse cruzamento.

A aplicação do operador cruzamento ocorre segundo uma probabilidade P_c. Sendo assim, é fixada uma taxa de probabilidade de cruzamento. Em cada iteração, é gerado aleatoriamente um valor, caso o valor seja menor ou igual à taxa, ocorrerá o cruzamento, caso contrário, não haverá cruzamento, e nesse caso os cromossomos descendentes serão cópias dos pais selecionados. O cruzamento simples ou de um ponto é usado neste trabalho. O operador cruzamento escolhe aleatoriamente um ponto de cruzamento, e a partir desse ponto é realizado o cruzamento de subseqüências dos cromossomos pais. No entanto, para assegurar a viabilidade dos novos descendentes, seu resultado deve ser combinado com a estratégia de mapeamento (ZHANG & GEN, 2006).

Na Figura 3, apresentamos um exemplo da aplicação do operador cruzamento simples. Duas soluções (pais) $P_1 = \{1,2,3,4,5,6,7,8,9,10\}$ e $P_2 = \{4,5,7,1,3,9,2,6,8,10\}$ são escolhidas, assim como o ponto de cruzamento igual a 4. Isto quer dizer, que o primeiro descendente D_1 tem os 4 primeiros elementos de P_1 e os elementos restantes são fornecidos por P_2 segundo a estratégia de mapeamento. O descendente D_2 tem os 4 primeiros elementos de P_2 e os elementos restantes fornecidos por P_1 segundo a estratégia de mapeamento. A estratégia de mapeamento consiste em preencher os elementos restantes dos descendentes seguindo a ordenação do outro pai de tal forma que a nova sequência gerada não contenha elementos repetidos, isto é, seja uma permutação, tal como $D_1 = \{1,2,3,4,5,7,9,6,8,10\}$ e $D_2 = \{4,5,7,1,2,3,6,8,9,10\}$.

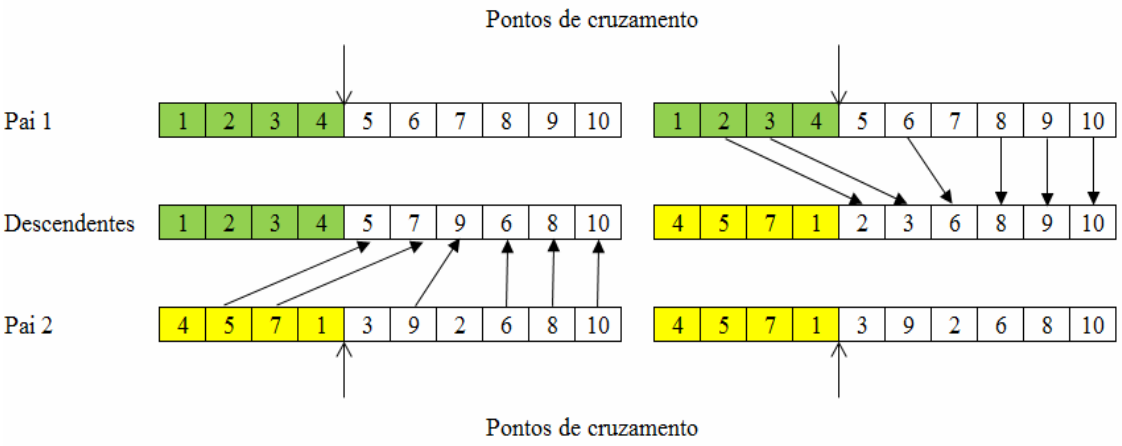

Figura 2. Exemplo de cruzamento simples para o escalonamento *flow shop*.

4.6 Mutação

O operador de mutação gera uma modificação arbitrária em uma solução candidata. É responsável pela diversidade dos indivíduos da população.

A aplicação do cruzamento ocorre segundo uma probabilidade P_m. Sendo assim, é fixada uma taxa de probabilidade de mutação. Em cada iteração, é gerado aleatoriamente um valor, caso o valor seja menor ou igual à taxa de mutação, ocorrerá a mutação, caso contrário, não haverá mutação, e nesse caso a solução candidata descendente será uma cópia da solução candidata logo após o cruzamento. Neste trabalho, a mutação é realizada escolhendo-se duas posições (tarefas) aleatoriamente e trocando suas posições na sequência (ZHANG & GEN, 2006).

Na Figura 4 apresentamos um exemplo de aplicação do operador mutação. A solução candidata $D = \{1,2,3,4,5,7,9,6,8,10\}$ é escolhida, assim como os pontos de mutação referentes às posições 3 e 8, correspondentes às tarefas 3 e 6, respectivamente. Então, trocando-se o conteúdo das posições, obtém-se em uma nova solução candidata dada por $DM = \{1,2,6,4,5,7,9,3,8,10\}$.

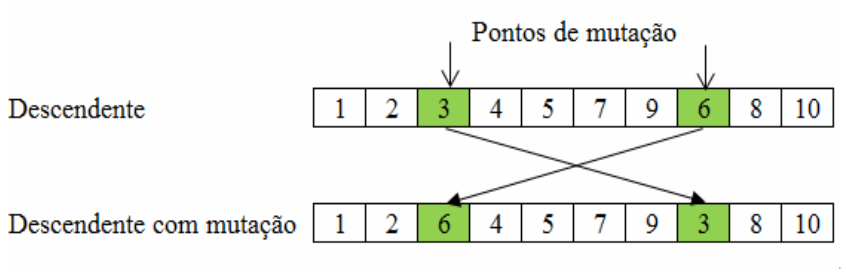

Figura 3. Exemplo de mutação por troca de tarefas no escalonamento *flow shop*.

5. Experimentos numéricos

Para os experimentos numéricos foram usados dados disponíveis na biblioteca pública OR-Library (2010), onde existem muitas instâncias do problema. O algoritmo genético, descrito acima, foi programado em MATLAB 6.0 e executado em um computador com processador de 2,4 GHz e 2GB de memória RAM.

No primeiro experimento numérico com o algoritmo genético, usamos a instância *car1* que contém tempos de processamento para o problema de escalonamento *flow shop* com 11 tarefas e 5 máquinas. A função aptidão escolha é $a_1(x)$ com w_1 e w_2 aleatórios em [0,1]. Os parâmetros utilizados no algoritmo genético foram: N_{gen} = 200, N_{pop} = 10, $N_{elite} = 3$, P_c = 0,9 e P_m = 0,3.

A Tabela 1 mostra as soluções (sequências) não-dominadas encontradas e os valores dos objetivos $C_{\max}$ e $\overline{F}$ destas soluções, depois de 10 execuções. Na Figura 4 se observa a aproximação da fronteira de Pareto.

5.1 Comparação em cenários

Aqui, vamos analisar 6 distintos cenários para a instância *car1* e *rec07* para conhecer o desempenho do algoritmo genético multiobjetivo em relação à escolha da função aptidão (existem 3 possibilidades) e dos valores de w_1 e w_2 (existem 2 possibilidades).

A Tabela 2 mostra os resultados encontrados para diferentes cenários depois de 10 execuções. Os valores médios dos objetivos $C_{\max}$, $\overline{F}$ e o número de soluções não-dominadas *ND* pertencentes ao conjunto de soluções acumuladas Pareto-ótimas em todas as gerações para todos os cenários.

Tabela 1. Soluções não-dominadas para a instância *car1*.

Sequência	C_{max}	$\overline{F}$
8 - 3 - 1 - 7 - 9 - 4 - 11 - 6 - 2 - 5 - 10	7161,00	4463,64
8 - 3 - 1 - 7 - 9 - 2 - 4 - 10 - 5 - 6 - 11	7963,00	4392,36
8 - 3 - 1 - 7 - 9 - 4 - 5 - 11 - 10 - 6 - 2	7038,00	4550,64
8 - 3 - 1 - 7 - 9 - 4 - 5 - 10 - 11 - 6 - 2	7047,00	4486,82
8 - 3 - 1 - 7 - 9 - 4 - 11 - 6 - 2 - 10 - 5	7528,00	4427,09
8 - 3 - 1 - 7 - 9 - 2 - 4 - 5 - 10 - 6 - 11	7746,00	4417,82
8 3 1 - 7 - 9 4 - 11 - 2 - 10 - 6 - 5	7395,00	4452,36
8 3 1 - 7 - 9 2 - 4 - 10 - 5 - 11 - 6	7454,00	4437,73
8 3 1 - 7 - 9 2 - 4 - 10 - 6 - 5 - 11	7994,00	4376,00

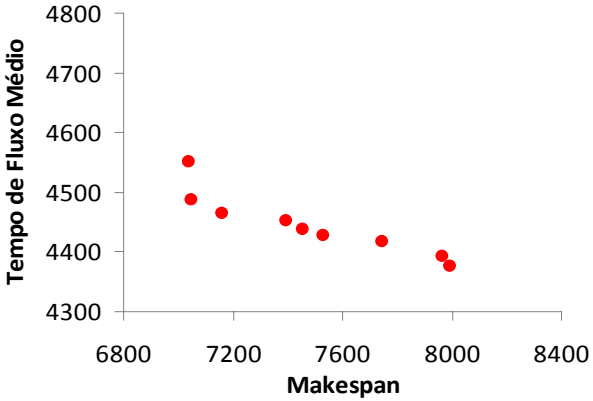

Figura 4. Aproximação da fronteira de Pareto para a instância *car1*.

Tabela 2. Resultados para diferentes cenários e valores de N_{gen} e N_{pop} para as instâncias *car1* e *rec07*

			a_1						a_2						a_3					
			$w_1 = w_2 = 0{,}5$			w_1 e w_2 aleatórios			$w_1 = w_2 = 0{,}5$			w_1 e w_2 aleatórios			$w_1 = w_2 = 0{,}5$			w_1 e w_2 aleatórios		
Instância	N_{gen}	N_{pop}	C_{max}	$\overline{F}$	*ND*	C_{max}	$\overline{F}$	*ND*	C_{max}	$\overline{F}$	*ND*	C_{max}	$\overline{F}$	*ND*	C_{max}	$\overline{F}$	*ND*	C_{max}	$\overline{F}$	*ND*
			Cenário 1			Cenário 2			Cenário 3			Cenário 4			Cenário 5			Cenário 6		
car1	100	10	7.684,4	4.504,8	0	7.324,3	4.523,9	0	7.360,3	4.486,9	3	7.228,8	4.510,8	0	7.505,6	4.616,6	0	7.231,8	4.496,7	0
($n = 11$,	200	10	7.550,5	4.432,3	3	7.480,7	4.444,9	1	7.329,4	4.542,9	0	7.271,2	4.505,2	0	7.491,4	4.447,0	1	7.255,3	4.458,3	1
$m = 5$)	500	20	7.249,3	4.492,6	1	7.491,4	4.435,4	2	7.313,7	4.440,6	7	7.289,4	4.446,2	8	7.331,3	4.452,3	4	7.399,5	4.450,6	3
			Cenário 1			Cenário 2			Cenário 3			Cenário 4			Cenário 5			Cenário 6		
rec07	100	10	1.684,3	1.076,5	0	1.690,8	1.088,5	0	1.680,4	1.087,1	0	1.697,0	1.086,2	0	1.612,8	1.088,3	0	1.642,0	1.088,2	0
($n = 20$,	200	10	1.636,3	1.087,1	0	1.632,6	1.095,7	0	1.608,3	1.099,1	0	1.695,5	1.074,1	0	1.695,5	1.072,4	2	1.653,8	1.100,5	0
$m = 10$)	500	20	1.608,5	1.082,2	0	1.663,2	1.081,3	0	1.619,0	1.080,6	0	1.653,9	1.081,3	1	1.612,4	1.086,7	0	1.609,0	1.080,6	2

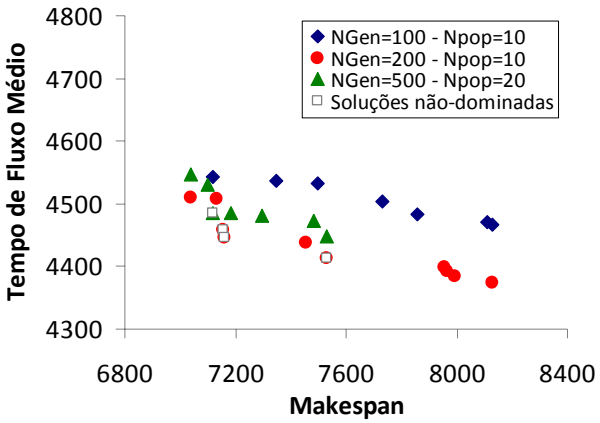

Figura 5. Resultados para a instância *car1* no cenário 1.

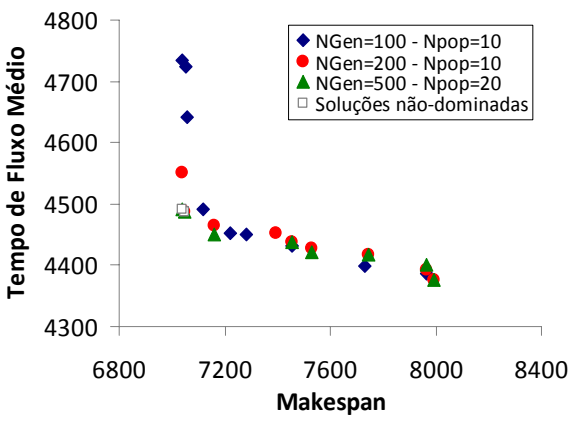

Figura 6. Resultados para a instância *car1* no cenário 2.

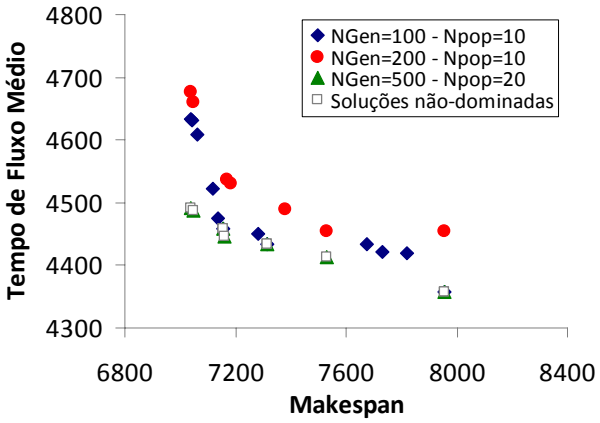

Figura 7. Resultados para a instância *car1* no cenário 3.

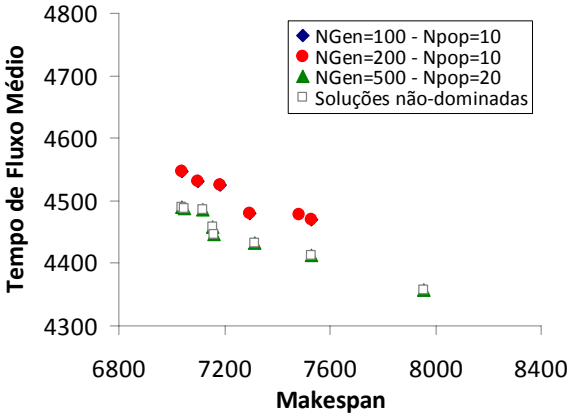

Figura 8. Resultados para a instância *car1* no cenário 4.

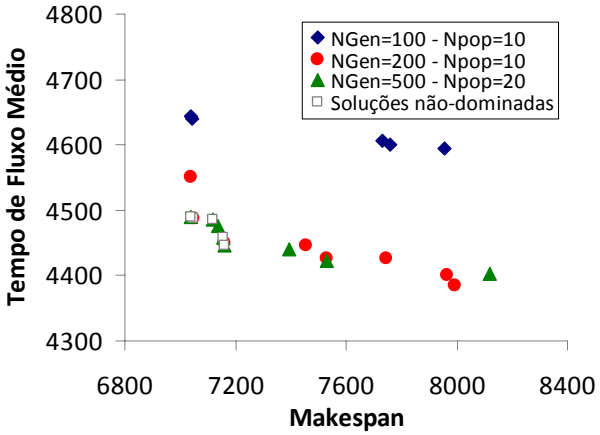

Figura 9. Resultados para a instância *car1* no cenário 5.

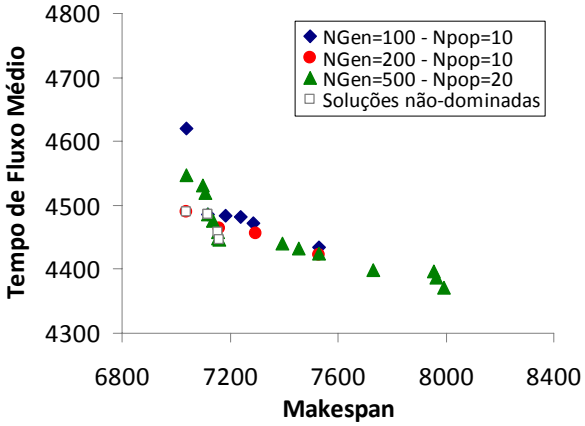

Figura 10. Resultados para a instância *car1* no cenário 6.

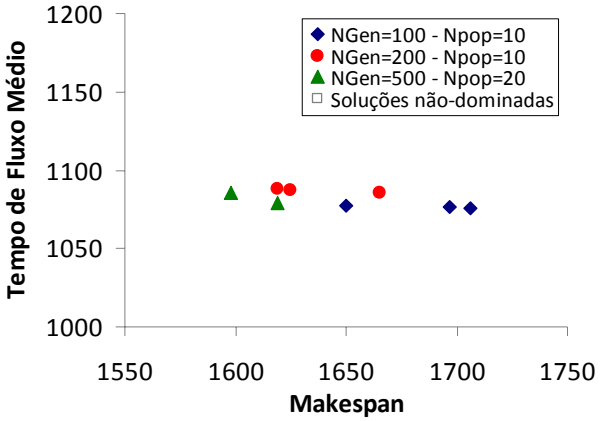

Figura 11. Resultados para a instância *rec07* no cenário 1.

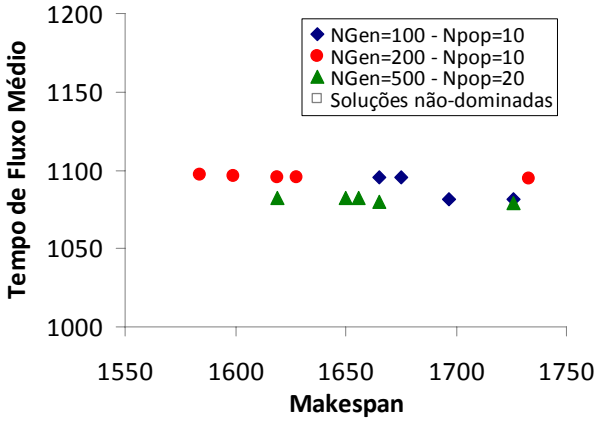

Figura 12. Resultados para a instância *rec07* no cenário 2.

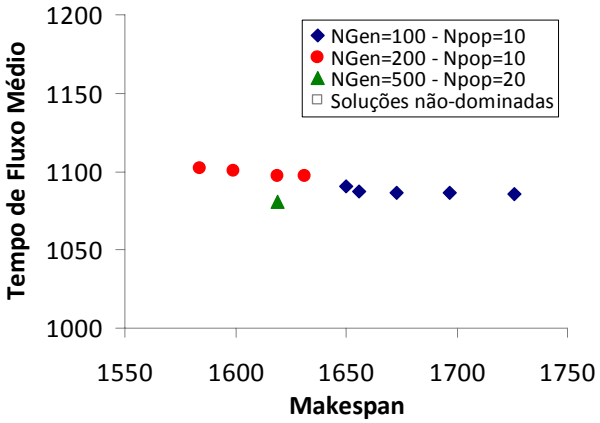

Figura 13. Resultados para a instância *rec07* no cenário 3.

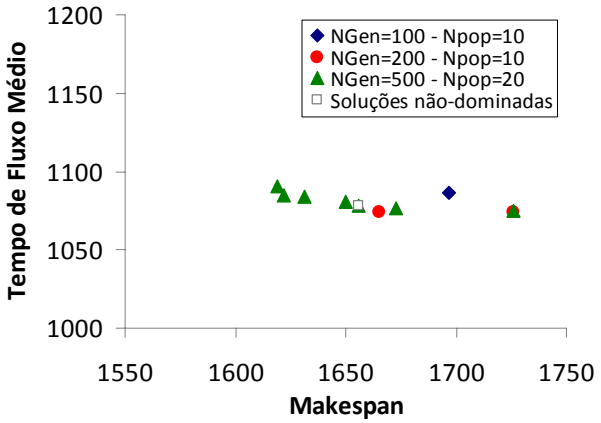

Figura 14. Resultados para a instância *rec07* no cenário 4.

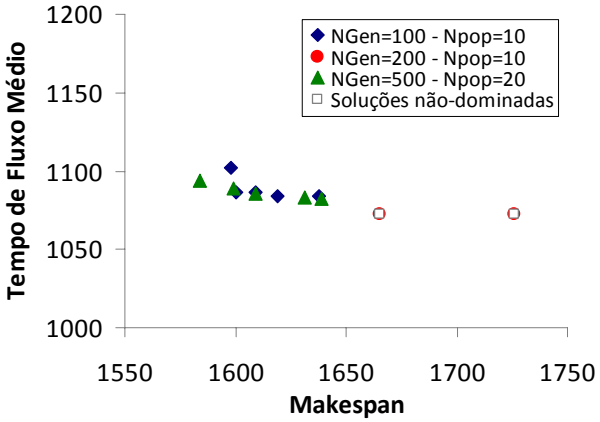

Figura 15. Resultados para a instância *rec07* no cenário 5.

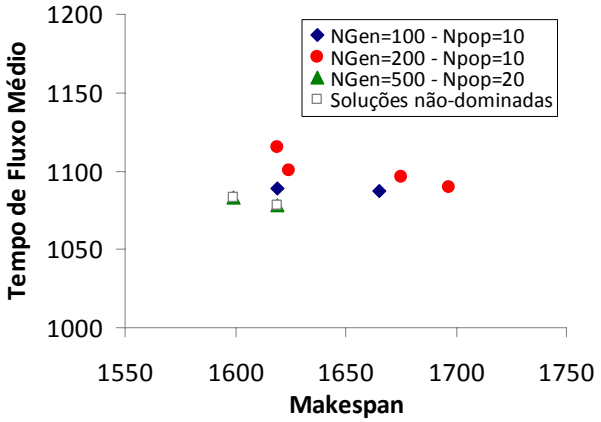

Figura 16. Resultados para a instância *rec07* no cenário 6.

Juntando os resultados de todos os cenários com distintos valores de N_{gen} e N_{pop}, tem-se para a instância *car1* um total de 8 soluções não-dominadas distintas e para a instância *rec07* um total de 5 soluções não-dominadas distintas. Os resultados dos experimentos numéricos indicam que para encontrar soluções que se aproximam à fronteira de Pareto é necessário um número elevado de gerações ($N_{gen} \geq 500$). Cabe ressaltar que nos cenários 1, 2, 5 e 6 da instância *car1* não se obteve muita diferença nos resultados com 200 ou 500 gerações. Já no cenário 3 foram obtidos 7 soluções não-dominadas e no cenário 4 foram obtidos todas as não-dominadas, observando que nestes cenários a função aptidão a_2 foi escolhida. Para a instância *rec07* nos cenários 1, 2 e 3, não foi possível obter pontos eficientes, somente nos outros cenários, porém em nenhum deles foi possível obter todas as soluções não-dominadas simultaneamente. Pode-se, também, apontar que, para as duas instâncias, nos cenários onde a função aptidão a_2 foi escolhida, foram obtidas boas soluções e que nos cenários onde a função aptidão a_3 foi escolhida foram obtidos mais pontos eficientes que nos cenários onde a função aptidão a_1 foi escolhida.

As figuras 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 e 16 ilustram o conjunto de soluções aproximadamente não-dominadas das instancias *car1* e *rec07* obtidas em cada cenário ao final de 10 execuções do algoritmo genético e também ilustram as soluções não-dominadas quando se juntam os resultados de todos os cenários com distintos valores de N_{gen} e N_{pop} de cada instância. Assim, é possível observar o nível de dispersão das soluções não-dominadas em relação à fronteira de Pareto. Os pontos de referência nos gráficos são as imagens das funções objetivo das soluções não-dominadas obtidas.

Nas Figuras 6, 8, 10, 12, 14 e 16 onde w_1 e w_2 são aleatórios, tem-se maior dispersão sobre a fronteira de Pareto que nas Figuras 5, 7, 9, 11, 13 e 15 onde $w_1 = w_2 = 0{,}5$, uma vez que, nas Figuras 6, 8, 10, 12, 14 e 16, a busca por soluções é multi-direcional, dada pela aleatoriedade de w_1 e w_2. Assim com w_1 e w_2 aleatórios encontramos mais soluções aproximadamente não-dominadas, permitindo, conseqüentemente, uma melhor aproximação do conjunto de Pareto-ótimo. Comparando os cenários com N_{gen}=100 e $w_1 = w_2 = 0{,}5$ em relação aos cenários com N_{gen}=100 e w_1 e w_2 aleatórios, verifica-se uma melhor aproximação da fronteira de Pareto no último caso, o que indica um maior desempenho na busca de soluções não-dominadas para este último caso. Nos cenários com N_{gen}=200 e N_{gen}=500 não se verifica diferença significativa entre as aproximações da fronteira de Pareto, no entanto, com N_{gen}=500 se tem melhor aproximação.

6. Conclusão

Neste trabalho, consideramos o problema de escalonamento *flow shop* com o objetivo de minimizar simultaneamente o *makespan* e o tempo de fluxo médio. Um algoritmo genético básico foi aplicado, com operadores de seleção por torneio, cruzamento simples e mutação com troca de conteúdos no cromossomo, e estratégia de elitismo, considerando ainda a possibilidade de uso de três funções aptidão distintas e duas formas diferentes de busca por soluções, com a finalidade de encontrar uma aproximação do conjunto ótimo de Pareto.

Um estudo da sensibilidade do algoritmo em gerar aproximadamente soluções não-dominadas ou eficientes foi realizado. A análise permitiu identificar a relação existente entre a escolha da função aptidão, a atribuição dos pesos que a compõem, a quantidade de pontos eficientes e de sua dispersão em relação a fronteira de Pareto.

Embora os experimentos numéricos tenham sido realizados considerando somente duas

instâncias do problema de escalonamento *flow shop*, eles podem indicar que função aptidão adotar e como atribuir valores para seus parâmetros dependendo do interesse ao se resolver o problema. Naturalmente, é necessário que o algoritmo genético básico seja aprimorado para resolver satisfatoriamente problemas de grande porte. Porém, temos interesse em resolver problemas típicos de um sistema de planejamento e programação avançado (APS) partindo do algoritmo genético desenvolvido e da experiência alcançada com esse estudo preliminar.

Referências

Buzzo, R. W., Moccellin, J.V. (2000), Programação da produção em sistemas flow shop utilizando um método heurístico híbrido algoritmo genético-simulated annealing, *Gestão & Produção*, 7, 364-377.

Ho, J. C. (1995),Flowshop sequencing with mean flowtime objective, *European Journal of Operational Research*, 81, 571-578.

Kamiri, N., Zandieh, M., Karamooz, H.R. (2010), Bi-objective group scheduling in hybrid flexible flowshop: A multi-phase approach, *Expert Systems with Applications*, 37, 4024-4032.

Konaka, A., Coitb, D.W., Smith, A.E. (2006), Multi-objective optimization using genetic algorithms: A tutorial, *Reliability Engineering and System Safety*, 91, 992-1007.

Liu, D., Yan, P., Yu, J. (2009), Development of a multiobjective GA for advanced planning and scheduling problem, *The International Journal of Advanced Manufacturing Technology*, 42, 974–992.

Liu, H., Abraham, A., Hassanien, A.E. (2010), Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm, *Future Generation Computer Systems*, 26, 1336 –1343.

Murata, T., Ishibuchi, H. (1995), MOGA: Multi-objective genetic algoritms, In *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, 289–294.

OR-Library, Welcome to OR-Library, 2010, acesso em junho de 2011. Disponível em http://people.brunel.ac.uk/~mastjjb/jeb/info.html .

Pasupathy, T., Rajendran, C., Suresh, R.K. (2006)**,** A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs, *The International Journal of Advanced Manufacturing Technology*, 27, 804-815.

Pinedo, M., *Scheduling: theory, algorithms, and systems*, Prentice Hall, New Jersey, 2008.

Ponnambalam, S. G., Jagannathan, H., Kataria, M., Gadicherla, A. (2004). A TSP-GA multi-objective algorithm for flow-shop scheduling, *The International Journal of Advanced Manufacturing Technology*, 23, 909- 915.

Reeves, C. R. (1995), A genetic algorithm for flowshop sequencing, *Computers & Operations Research*, 22, 5-13.

Widmer, M. e Hertz, A. (1989), A new heuristic method for the flow shop sequencing problem, *European Journal of Operational Research*, 41, 186-193.

Yang, J., Tang, W. (2009), Preference-based adaptive genetic algorithm for multiobjective advanced planning and scheduling problem, *International Conference on Industrial Engineering and Engineering Management*, number 5373213, 1935-1940.

Zhang, H., Gen, M. (2006), Effective genetic approach for optimizing advanced planning and scheduling in flexible manufacturing system. *In Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 17, 1841–1848.