UM ALGORITMO EVOLUCIONÁRIO APLICADO AO PROBLEMA *FLOWSHOP* PERMUTACIONAL COM RESTRIÇÃO DE ESPERA

Francisco Régis Abreu Gomes
Pós-Graduação em Logística e Pesquisa Operacional-UFC
Campus do Pici, Bloco 703, CEP 60455, Fortaleza-CE
regis@glen.ufc.br

José Lassance de Castro Silva
Universidade Federal do Ceará-UFC
Campus do Pici, Bloco 910, CEP 60455-760, Fortaleza-CE
lassance@pesquisador.cnpq.br

Resumo

Este trabalho aborda uma técnica de resolução para o Problema de Seqüenciamento de tarefas sem restrição de espera, denominado na literatura de Continuous Permutation Flowshop Scheduling Problem (C*PFSP*), que possui a restrição de que nenhuma tarefa pode esperar por processamento entre máquinas consecutivas. Foram resolvidos dois tipos de CPFSP, um com a função objetivo sendo o tempo total de fluxo e o outro sendo o *makespan*. A técnica de resolução é baseada na metaheurística Algoritmo Genético (*AG*) que tem sido aplicada com sucesso aos problemas da classe *Permutation Flowshop Scheduling Problem* (PFSP). O AG não utilizou inicialização eficiente e/ou hibridização, com uma técnica de busca. Foram adotados critérios para a diversificação e intensificação na busca por boas soluções para o problema. Vários experimentos computacionais foram realizados e comparados com outros métodos encontrados na literatura, onde foi possível constatar o bom desempenho do método desenvolvido.
Palavras-chave: Problema de Seqüenciamento, Algoritmo Genético, Metaheurística.

Abstract

This work approaches a resolution technique for the Continuous Permutation Flowshop Scheduling Problem (CPFSP) without wait for processing among consecutive machines. Two types of CPFSP were resolved. One with the function objective being the total time of flow and the another being the makespan. The resolution technique is based on the metaheurística Genetic Algorithm (AG) that has been applied with success to the problems of the class Permutation Flowshop Scheduling Problem (PFSP). AG didn't use initialization efficient and/ou hibridization, with a search technique. Criterias were adopted for the diversification and intensification in the search for good solutions of the problem. Several experiments computacionais were accomplished and compared with other methods found in the literature, where it was possible to verify the good acting of the developed method.
Key-words: Scheduling Problem, Genetic Algorithm, Metaheuristic.

1. Introdução

O Problema de Seqüenciamento Permutacional Contínuo *Flowshop*, denominado na literatura de *Continuous Permutation Flowshop Scheduling Problem* (CPFSP), é um problema de programação de operações em um ambiente *Permutation Flowshop* onde *n* tarefas devem ser processadas por um conjunto de *m* máquinas distintas, tendo o mesmo fluxo de processamento nas máquinas. Sua principal característica é a restrição de que uma tarefa tendo sido iniciada não pode ficar esperando por processamento entre máquinas consecutivas, ou seja, o fluxo de processamento das *n* tarefas no conjunto das *m* máquinas deve ser contínuo. Usualmente a solução do CPFSP consiste em determinar uma seqüência de tarefas dentre as *n*! seqüências possíveis que otimize uma função objetivo estabelecida.

Segundo Hall e Sriskandarajah (1994) existem duas razões para a ocorrência do CPFSP. A primeira razão é a tecnologia de produção empregada, por exemplo, a temperatura ou outra característica de um material requer que cada operação siga imediatamente para a próxima etapa. Essas situações são comuns nas indústrias siderúrgica, química, farmacêutica, alimentícia e plástica. Ambientes de manufatura moderna como *just-in-time*, sistemas flexíveis de manufatura e células robóticas exigem uma complexa coordenação no processo de manufatura. Isto também pode ocorrer em empresas de serviço onde o custo de atendimento do cliente é alto. A segunda razão de ocorrência é a falta de espaço de estocagem intermediária, isto ocorre geralmente em linhas de produção automáticas e em sistemas de estoque gerenciado por *kanbans* (cartões informando a quantidade de produtos a serem produzidos), pois nestes casos o estoque em processo tem uma quantidade fixa limitada.

No CPFSP são dados(as):

a) um conjunto com *n* tarefas $J_1, J_2,..., J_n$;
b) um conjunto com *m* máquinas $M_1, M_2, ..., M_m$;
c) *m* operações, com uma operação representando o tempo de processamento da tarefa por máquina;
d) o fluxo de operações nas máquinas é para qualquer j=1,2,..., n, a tarefa J_j deve ser processada primeiro na máquina M_1, depois na máquina M_2, e assim por diante até a última máquina, no caso máquina M_m;
e) O processamento das tarefas não possui folga entre máquinas consecutivas, isto é, para qualquer j =1,2,..., n, a tarefa J_j deve ser processada primeiro na máquina M_1, depois imediatamente (sem folga) na máquina M_2, e assim por diante até a última máquina, no caso máquina M_m;
f) Uma máquina pode processar somente uma operação de cada vez, e iniciada uma operação, ela deva ser processada até a sua conclusão;

O *input* do CPFSP é dado por *n*, *m* e uma matriz $P(n \times m)$ de elementos não negativos, onde P_{ij} denota o tempo de processamento da tarefa J_j na máquina M_i. Hall e Sriskandarajah (1994) apresentam a classificação de vários problemas do tipo CPFSP usando a recente classificação paramétrica α/β/γ, proposta por Graham *et al.* (1979). Esse trabalho aborda dois problemas do tipo CPFSP cuja única diferença está na função objetivo de cada um. O primeiro problema usa como função objetivo o *makespan* da seqüência de tarefas, sendo classificado como F/no-wait/C_{max} e o segundo problema usa como função objetivo o tempo total de fluxo da seqüência de tarefas, ou seja, a soma do tempo de complementação de todas as tarefas da seqüência, sendo classificado como F/no-wait/$\sum C_j$. O CPFSP pertence à classe dos problemas NP-*hard*, quando $m \geq 3$, conforme Röck (1984), de forma que pode ser resolvido eficientemente de maneira ótima somente em casos de pequeno porte.

Segundo Silva e Soma (2006a) um Problema de Otimização Combinatória Permutacional (POCP) pode ser definido por um terno (S, g, *ins*), onde S é o conjunto de todas as soluções do problema, g é sua função ou procedimento que associa a cada solução viável $s \in S$ um número real e *ins* é uma instância do problema. O número de soluções existentes para um POCP é representado por |S| (cardinalidade de S) e igual a $n!$ (fatorial de n). O objetivo é encontrar uma solução $s^* \in S$ que otimize a um dado critério de desempenho representado pela função g.

Representa-se s como uma permutação de n elementos, ou seja, s=<a_1, a_2, ..., a_n>, com $a_i \neq a_j$, $\forall$ $1 \leq i, j \leq n$ e $i \neq j$.

O CPFSP pode ser modelado como um POCP da seguinte forma:

a) Um elemento s=<J_1, J_2, ..., J_n>, do conjunto de soluções viáveis S é representado por uma permutação das n tarefas, com a ordem de s determinando a seqüência na qual as tarefas serão processadas; e

b) O procedimento g, que atribui um valor real para cada solução viável do problema, é dado abaixo pelas equações 1 e 2, para cada tipo de problema com sua função objetivo, conforme Fink e Voβ (2003). A seguir são apresentadas as duas formas analíticas para calcular o valor de g de cada uma das funções objetivo.

Tempo Total de Fluxo (TTF): $g_{TTF} = \sum_{i=2}^{n}(n+1-i)d_{s(i-1),s(i)} + \sum_{i=1}^{n}\sum_{j=1}^{m}P_{ij}$ (1).

Makespan : $g_{makespan} = \sum_{i=2}^{n}(n+1-i)d_{s(i-1),s(i)} + \sum_{j=1}^{m}P_{s(n)j}$ (2).

A variável d_{iy} usada nas Equações 1 e 2 apresentadas acima é o tempo de espera na primeira máquina entre o começo da tarefa J_i e o começo da tarefa J_y, quando a tarefa J_y é processado diretamente após a tarefa J_i, com $1 \leq i \leq n$ e $1 \leq y \leq n$, com $i \neq y$. A Equação 3, dada abaixo, mostra como calcular o valor de d_{iy}.

$$d_{iy} = \max_{1 \leq j \leq m}\{\sum_{h=1}^{j}P_{ih} - \sum_{h=2}^{j}P_{y,h-1}\} \quad (3).$$

A metaheurística Algoritmo Genético (AG) baseada na evolução das espécies, já foi utilizada na resolução do CPFSP, exemplos são os trabalhos de Chen *et al.* (1996), Aldowaisan e Allahverdi (2003) e Schuster e Framinan (2003). Os AGs de Chen *et al.* (1996) e Aldowaisan e Allahverdi (2003) foram testados em problemas gerados aleatoriamente o que torna difícil a comparação com outros métodos que usaram dados da OR-Library. Somente o AG de Schuster e Framinan (2003) foi testado em instâncias conhecidas. Fink e Voβ (2003) desenvolveram várias heurísticas e alguns métodos baseados nas metaheurísticas *Simulated Anneling* e *Tabu Search* para o CPFSP que foram testados nas instâncias de Taillard (1993). Grabowski e Pempera (2005) desenvolveram um método baseado na metaheurística *Tabu Search* para o CPFSP que obteve um resultado melhor que o AG de Schuster e Framinan (2003).

É uma característica dos AG aplicados ao CPFSP à utilização das seguintes estratégias: inicialização eficiente ou uma etapa de hibridização com uma técnica de busca. Gomes e Silva (2007) mostraram para o PFSP que o seu AG, que tenta diversificar e intensificar o processo de busca e não recorre a nenhuma das duas estratégias mencionadas anteriormente, obteve resultados computacionais melhores que os AGs de Silva e Soma (2006b) e Chen *et al.* (1995) que usaram alguma dessas estratégias.

Até o momento não foi encontrada, na literatura, informação sobre a eficiência de um AG tradicional aplicado ao CPFSP. Os AGs pesquisados recorrem aquelas duas estratégias descritas anteriormente Assim sendo, propomos nesse trabalho desenvolver um AG tradicional na resolução do CPFSP, que não recorra a nenhuma dessas estratégias. O AG desenvolvido será denominado de *rAG* para diferenciar dos outros AG existentes na literatura. Os problemas de teste utilizados para o F/no-wait/C_{max} foram as instâncias de Reeves (1995) e Heller (1960) e para o F/no-wait/$\sum C_j$ foram as instâncias de Taillard (1993).

O restante desse artigo é organizado como segue. Na Seção 2 é apresentada a descrição do rAG. Na Seção 3 descrevemos os resultados obtidos com os experimentos computacionais realizados com o rAG e a comparação com outros métodos. Na Seção 4 são apresentadas às conclusões do trabalho, enquanto que, na Seção 5 é apresentado o material bibliográfico consultado.

2. Descrição do rAG

O AG foi criado por John Holland durante as décadas de 1960 e 1970 (Holland, 1975). Segundo Haupt e Haupt (2004), o AG é uma técnica baseada nos princípios da genética e seleção natural das espécies. A técnica é formada por uma população de indivíduos que representam as soluções do problema. Cada indivíduo da população é avaliado segundo sua qualidade em relação aos outros indivíduos da população. Os indivíduos são escolhidos por um procedimento inspirado na seleção natural para passarem por operações genéticas que resultam em descendentes que comporão a nova população. Os estudos mostram que a nova população tem a tendência de ter indivíduos com aptidões melhores que os indivíduos da população anterior. Este processo de gerar novas populações é chamado de geração. O melhor indivíduo da última população associado a uma solução do problema é selecionado como a melhor solução encontrada para o problema.

O desenvolvimento do rAG levou em consideração as características que fariam a evolução da qualidade das soluções agirem de forma melhor e por mais tempo mesmo sem inicialização eficiente e/ou hibridização. Para isso foram usados dois princípios para guiar a construção do rAG. Mitchell (1998) afirmou que no AG a evolução das soluções depende da variação nas aptidões dos indivíduos da população, ou seja, da diversificação das soluções. A intensificação no processo de busca tende a melhorar a qualidade das soluções finais encontradas (Silva e Soma, 2001; Grabowski e Pempera, 2005; Dréo *et al.*, 2006). Daí se escolheu a diversificação e a intensificação como características importantes para a qualidade do rAG.

Depois do desenvolvimento do primeiro rAG, seguindo o modelo tradicional, foram desenvolvidos e testados três procedimentos baseados nos princípios da diversificação e intensificação para melhorar o seu desempenho. O *primeiro procedimento* é baseado no princípio da diversificação e consiste em permitir, na etapa da formação da nova população do rAG, que indivíduos de aptidão menor, mas com características diferentes de todos os outros indivíduos da população tenham chance de serem escolhidos para a nova população. O *segundo procedimento* é baseado no princípio da intensificação e consiste em fazer o melhor indivíduo da população passar por um processo genético com outros indivíduos da população mais vezes que o comum. Por fim, o *terceiro procedimento* é baseado no princípio da diversificação e consiste em realizar uma perturbação em todos os indivíduos da população depois que um estado de estagnação é identificado. A seguir descrevemos as 8 principais componentes do rAG.

2.1. *Método de representação das soluções*

O CPFSP nesse trabalho está sendo modelado como um POCP. Logo uma permutação de tarefas representa uma solução do problema sendo a ordem relativa das tarefas na permutação indicando a ordem de processamento das mesmas.

2.2. *Método de geração da população inicial*

Como o rAG não adota inicialização eficiente, o método de geração dos indivíduos da população inicial é aleatória, na verdade pseudo-aleatória, por causa dos algoritmos de geração de números aleatórios usados pelos programas de computador. Para eliminar os indivíduos repetidos da população inicial, foi implementado um procedimento que consiste em analisar todos os indivíduos da população e em encontrando um indivíduo repetido fazer ele passar pelo processo de mutação, descrito adiante, até que se torne um indivíduo de seqüência única na população inicial.

2.3. *Método de cálculo da função de aptidão dos indivíduos*

Para calcular a aptidão dos indivíduos do problema F/no-wait/$\sum C_j$ foi usada a Equação 1 e para calcular a aptidão dos indivíduos do problema F/no-wait/C_{max} foi usada a Equação 2.

2.4. *Método de seleção dos indivíduos*

O rAG usou o método de seleção por torneio, *cf.* Mitchell (1998), é resistente a convergência prematura e tem custo computacional baixo. O método de seleção foi implementado da seguinte

forma: no início da execução do rAG deve ser especificado um parâmetro de controle *k*, entre 0 e 1, que será o mesmo durante toda a execução; O procedimento inicia com a escolha de um sub-conjunto de indivíduos da população atual de tamanho *d*, em seguida é gerado aleatoriamente um número entre 0 e 1, então se esse número for menor que *k*, o melhor indivíduo do sub-conjunto é escolhido, se não o segundo melhor indivíduo do sub-conjunto é o escolhido.

Nesta etapa, o segundo procedimento proposto para melhorar o desempenho do rAG foi o da seleção elitista e funciona da seguinte forma: é definido no início da execução do rAG um valor maior que 0 e menor que 1 para o parâmetro P_{ce}, o procedimento se inicia com a geração de um número aleatório entre 0 e 1, então se esse número for maior que o P_{ce} os dois pais são selecionados pela seleção por torneio, se não um dos pais será o melhor indivíduo da população atual e o outro pai será selecionado pela seleção por torneio. O objetivo deste procedimento é favorecer o processo de intensificação, já que o processo de *crossover* realizado com o melhor indivíduo da população tem grandes chances de gerar descendentes com boa aptidão.

2.5. *Operadores genéticos*

O operador *crossover* usado foi o *Order Crossover* (OX) (Goldberg, 1989). O operador de mutação implementado foi o movimento *swap*. Nesta etapa foi implementado o terceiro procedimento proposto para melhorar o desempenho do rAG. O novo operador genético é chamado de mutação populacional. Estes três operadores genéticos são descritos a seguir. Na aplicação dos operadores genéticos foi adotada a estratégia *crossover*-ou-mutação, i.e., sempre um dos operadores é aplicado, ou *crossover* ou mutação, mas não ambos.

O operador *crossover* OX foi criado baseado na idéia dos bons blocos construídos. Por isso, baseia-se nas posições relativa e absoluta das tarefas na seqüência. Este procedimento é descrito a seguir.

1 – São escolhidos dois pais através do método de seleção;

2 – É escolhido aleatoriamente o mesmo fragmento de cada um dos pais e copiado nos respectivos filhos (Figura 1). Esta etapa preserva as posições absoluta e relativa das tarefas na seqüência; e

3 – As posições não-preenchidas de cada filho são copiadas das tarefas do outro pai no sentido da esquerda para a direita (Figura 1). Este procedimento faz com que seja preservada a ordem relativa das tarefas na seqüência.

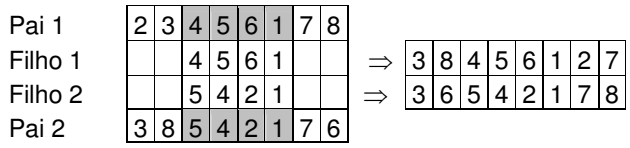

Figura 1 – Segunda e terceira etapa do *crossover* OX.

Mutação

O operador mutação *swap* consiste em realizar uma única alteração na estrutura do indivíduo. Este operador é implementado da seguinte forma: são escolhidos aleatoriamente duas posições na estrutura do indivíduo e os valores dessas posições são trocados (permutados). Nesta etapa também existe a preocupação de evitar que surja na população um indivíduo repetido, por isso, quando o indivíduo gerado já existe na população o procedimento é repetido até a sua estrutura ser a única.

Mutação Populacional

A mutação populacional é baseada no princípio da diversificação e foi o terceiro procedimento proposto para melhorar o desempenho do rAG a ser implementado. A mutação populacional consiste em realizar a mutação *swap* em todos os indivíduos da população depois de um determinado número de gerações sucessivas sem melhoria, denominado G_e, ter sido atingido.

Ainda nesta etapa, quando um indivíduo gerado é repetido ele sofre mutação novamente até sua seqüência ser única na população. Este procedimento é descrito a seguir.

Passo 1 : Se $f_{0i} = f_{0(i-1)}$, então $c := c+1$, caso contrário $c := 0$;
Passo 2 : Se $c = G_e$, então:
Se $f_{0i} < f^*$, então $\{s^* := s_{0i}$ e $f^* := f_{0i}\}$,
Se não $\{s_{0i} := s^*$; todos os indivíduos da população sofrem mutação *swap*; $c := 0;\}$
Onde: f^* : é o valor da melhor aptidão em todas as gerações já realizadas;
s^* : é o melhor indivíduo em todas as gerações já realizadas;
f_{0i} : é o valor da melhor aptidão na população i;
s_{0i} : é o melhor indivíduo da população i;
c : é o número de gerações sucessivas sem melhoria de f_{0i};
i : é o número da *i-ésima* geração; e
G_e : número de gerações sucessivas sem melhoria.

2.6. *Método de estratégia geracional*
Este método é baseado na estratégia *population overlaps* que segundo Reeves e Rowe (2002) consiste em preservar uma parte da população atual chamada de *generation gap* e substituir a outra parte por novos indivíduos gerados pelos procedimentos genéticos para compor a nova população. Para evitar que existam indivíduos repetidos na população, um indivíduo só é aceito para ser incorporado na nova população se a sua estrutura não é repetida. A outra condição depende da aptidão do indivíduo. Se a aptidão do novo indivíduo f é melhor do que a pior aptidão da população f_p, então este indivíduo substitui o indivíduo de pior aptidão e o valor da pior aptidão é atualizado.

Nesta etapa o primeiro procedimento proposto, baseado no princípio da diversificação, para melhorar o desempenho do rAG foi implementado. A estratégia de só aceitar um indivíduo com aptidão melhor que a pior aptidão diminuía a diversidade da população e, por isso, foi implementada a possibilidade controlada pelo parâmetro P_a de um indivíduo com aptidão inferior a pior aptidão existente na população ser aceito, deste que tenha seqüência única em relação a população atual. Este procedimento é descrito a seguir:
Passo 1 : Execução do operador *crossover*;
Passo 2 : Se $f < f_p$ e $s \neq s_i$ ($\forall\, i = 1, 2, ..., N$), então: $s_p := s$ e atualiza f_p;
Passo 3 : Se $f \geq f_p$ e $s \neq s_i$ ($\forall\, i = 1, 2, ..., N$), então:
- gera-se um número aleatório r entre 0 e 1:
- se $r < P_a$, então: $s_p := s$ e atualiza f_p;
Onde:
N : é o tamanho da população;
P_a : probabilidade de aceitar um indivíduo;
s : é um dos indivíduos gerado pelo operador *crossover*;
f : é a aptidão de s;
s_p : é o indivíduo de pior aptidão; e
f_p : é a aptidão de s_p.

2.7. *Critério de parada*
O critério de parada utilizado pelo rAG foi o tempo de execução. Este critério de parada foi adotado por dois motivos: facilitar a comparação com os outros métodos, pois o tempo de execução é utilizado na comparação e as soluções da população do rAG não tendem a convergir rapidamente devido aos procedimentos *estratégia populacional* e *mutação populacional*.

2.8. *Determinação dos valores dos parâmetros do rAG*
A última etapa num projeto de um AG é a definição dos valores dos seus parâmetros, como tamanho da população, taxa de *crossover* e taxa de mutação. Segundo Mitchell (1998) os parâmetros de um AG interagem entre si de forma não-linear. Sendo assim, não podem ser

otimizados ao mesmo tempo. Muitos trabalhos têm sido realizados nesta área, entretanto nenhuma função matemática foi apresentada, que forneça os melhores valores para esses parâmetros (DeJong, 1980; Grefenstette, 1986 e Ruiz *et al.*, 2006). Por isso, uma desvantagem do AG é a dificuldade do processo de calibração dos seus parâmetros (Dréo *et al.*, 2006). Diante disso, foi criado até um AG que utiliza poucos parâmetros por Lobo e Goldberg (2004).

Foram realizados alguns testes para determinar os valores dos parâmetros do rAG, mas não foi utilizado nenhum procedimento sofisticado para isso. Durante a determinação dos valores dos parâmetros do rAG se percebeu que para o problema F/no-wait/$\sum C_j$ o tamanho da população e o parâmetro G_e maiores melhorava a qualidade das soluções. Assim, foram usados dois conjuntos de valores para os parâmetros do rAG, apresentados a seguir.

Os parâmetros do rAG para o problema F/no-wait/C_{max}: Tamanho da população (N) : 30; Tamanho do sub-conjunto de seleção (d) : 3; Parâmetro da seleção por torneio (k) : 0,7; Taxa de *crossover* (P_c) : 0,70; Taxa de aceitação (P_a) : 0,30; Taxa de *crossover* elitista (P_{ce}): 0,30; Taxa de mutação (P_m) : 0,05; e Gerações de estagnação (Ge) : 25.

Os parâmetros do rAG para o problema F/no-wait/$\sum C_j$: Tamanho da população (N) : 75; Tamanho do sub-conjunto de seleção (d) : 3; Parâmetro da seleção por torneio (k) : 0,7; Taxa de *crossover* (P_c) : 0,70; Taxa de aceitação (P_a) : 0,30; Taxa de *crossover* elitista (P_{ce}): 0,30; Taxa de mutação (P_m) : 0,05; e Gerações de estagnação (G_e) : 50.

3. Experimentos Computacionais

O código do rAG foi implementado em Delphi 7. Os experimentos foram realizados em um computador PC-AMD (2.2 GHz e 512 MB de RAM). Devido a natureza probabilística dos AGs tradicionais, o rAG foi executado cinco vezes para cada problema e escolhido o melhor resultado. O principal indicador utilizado nas comparações entre os métodos é o percentual de desvio das soluções, dado por $((z' - z^*) / z^*)\times100$, onde z^* é a melhor solução do problema e z' é a melhor solução encontrada pelo método de resolução aplicado ao problema.

3.1. Experimento 1 – etapas de melhoria do rAG

O primeiro experimento tem o objetivo de mostrar e analisar a melhoria obtida pelo rAG com a utilização dos procedimentos propostos baseados nos princípios da diversidade e intensificação no CPFSP. Nesta etapa foram realizados quatro tipos de experimentos. O primeiro experimento foi realizado com o chamado rAG-1 que não tem implementado nenhum dos três procedimentos propostos. O segundo experimento foi realizado com o rAG-1 acrescido do procedimento que permite que indivíduos com aptidão menor que a pior aptidão da população tenham probabilidade de serem aceitos na população, e foi denominado de rAG-2. O terceiro experimento foi realizado com o rAG-2 acrescido do procedimento seleção elitista, e foi denominado de rAG-3. O quarto e último experimento foi realizado com o rAG-3 acrescido do procedimento mutação populacional, e foi denominado de rAG-4. Foi utilizado o problema F/no-wait/$\sum C_j$ e as instâncias de Taillard (1993).

O resumo dos resultados desses experimentos com os tempos de execução usados são mostrados na Tabela 1. Nesta tabela a coluna um mostra as classes das instâncias de Taillard (1993), as colunas dois a cinco mostram o percentual de desvio para o rAG-1, rAG-2, rAG-3 e rAG-4 em relação as melhores soluções obtidas por Fink e Voβ (2003), respectivamente, e a coluna seis mostra o tempo de execução usado em cada classe de problemas.

A análise dos dados da Tabela 1 produz as seguintes observações:

a) O primeiro procedimento (rAG-2) passou o desvio do rAG de 1,710% para 0,517%, o segundo procedimento (rAG-3) passou o desvio de 0,517% para 0,174% e o terceiro procedimento (rAG-4) passou o desvio de 0,174% para 0,171%;
b) O acréscimo do último procedimento melhorou o desempenho do rAG em apenas 0,003%, isto se explica pelos baixos tempos de execução utilizados, pois a característica deste procedimento é agir quando a população se encontra em estado de estagnação e o que ocorre com menos freqüência com poucas gerações executadas;

c) A eficiência do terceiro procedimento é percebida nas instâncias com 20 tarefas porque estes problemas são menos complexos e, por isso, rapidamente o rAG encontra boas soluções e, por isso, a população entra em estado de estagnação e a mutação populacional passa a ter um papel ativo; e
d) O resultado do rAG-4 ter sido inferior ao resultado do rAG-3 para a classe 100x5 se deve a utilização dos números aleatórios gerados pelo Delphi 7, e como esses dois algoritmos têm quantidades de execuções de números aleatórios diferentes, isso influenciou no resultado.

Tabela 1 – Resumo da comparação das várias etapas de melhoria do rAG.

Instâncias (*n* x *m*)	rAG-1 (%)	rAG-2 (%)	rAG-3 (%)	rAG-4 (%)	Tempo (s)
20 x 5	0,55	0,15	0,12	0,04	0,08
20 x 10	0,39	0,11	0,04	0,02	0,08
20 x 20	0,51	0,11	0,08	0,05	0,08
50 x 5	0,67	0,22	0,15	0,14	3,75
50 x 10	0,70	0,29	0,18	0,15	3,75
50 x 20	1,01	0,40	0,21	0,12	3,75
100 x 5	1,83	0,59	-0,10	-0,01	10,00
100 x 10	2,68	0,71	0,18	0,18	10,00
100 x 20	2,86	1,03	0,51	0,34	10,00
200 x 10	3,28	0,79	0,00	0,16	50,00
200 x 20	4,33	1,28	0,53	0,68	50,00
Média	**1,710**	**0,517**	**0,174**	**0,171**	**15,96**

Estes resultados mostram que foi proveitosa a implementação dos procedimentos propostos para o desempenho do rAG. O rAG-4 foi o que obteve os melhores resultados, por isso, passará a ser chamado apenas de rAG e será o algoritmo usado daqui para frente.

3.2. Experimento 2 – comparação do rAG com o GASA e o Tabu Search-Multimovimento

Este experimento tem como objetivo analisar o desempenho do rAG no problema F/no-wait/C_{max}. Para isso o rAG foi testado com as instâncias de Reeves (1995) e Heller (1960). Os resultados do rAG são comparados com o algoritmo híbrido (*Genetic Algorithm + Simulated Anealing*) de Shuster e Framinan (2003), denominado de GASA, único AG encontrado na literatura para o CPFSP que utiliza instâncias conhecidas nos seus testes e o melhor método de Grabowski e Pempera (2005) que é um Tabu Search com Multimovimento (TS-M) e cuja a solução inicial é gerada pela heurística NEH de Nawaz *et al.* (1983) que é o melhor método encontrado na literatura para o problema F/no-wait/C_{max}. Os resultados do TS-M foram obtidos num computador Pentium 1.000 MHz. Os resultados do GASA foram obtidos num computador Athlon 1.400 MHz. Daí, os experimentos com o rAG utilizaram metade do tempo utilizado pelo GASA e pelo TS-M para tornar a comparação justa.

Estes dois métodos foram escolhidos para serem comparados com o rAG devido aos seus testes terem sido realizados com as instâncias de Reeves (1995) e Heller (1960). A Tabela 2 mostra os resultados do rAG e a comparação com o GASA, enquanto a Tabela 3 mostra a comparação com o TS-M. Nas Tabelas 2 e 3 a coluna um mostra o nome das instâncias, a coluna dois mostra o número de tarefas e máquinas das instâncias, a coluna três mostra os resultados do GASA ou do TS-M, a coluna quatro mostra os tempos usados pelo GASA ou pelo TS-M, a coluna cinco mostra os resultados do rAG, a coluna seis mostra o tempo de execução usado pelo rAG, a coluna sete mostra a razão entre o tempo de execução do rAG e do GASA ou do TS-M multiplicado por dois, devido a consideração do computador utilizado nos experimentos do rAG ser duas vezes mais rápido do que os computadores utilizados pelos outros métodos, e a coluna oito mostra o desvio entre o rAG e o GASA ou o TS-M.

A análise da Tabela 2 produz as seguintes observações:

a) Nos 23 problemas testados o rAG obteve melhor resultado em 21 problemas e em 2 problemas obteve resultado igual ao GASA. Na média o rAG foi 4,99 % superior ao GASA;

b) O tempo de execução do rAG para os 23 problemas sempre foi inferior ao utilizado pelo GASA. A média da razão entre os tempos usados pelo rAG e pelo GASA foi 0,014 o que significa que o rAG é mais rápido do que o GASA;
c) A menor razão entre os tempos usados pelo rAG e o GASA foi de 0,003 o que significa que o rAG foi 333 vezes mais rápido que o GASA e, a maior razão foi 0,033 o que significa que o rAG foi 30 vezes mais rápido que o GASA;
d) O melhor resultado do rAG em comparação com o GASA foi -16,76% no problema hel1;
e) Quando o rAG foi melhor que o GASA a menor diferença ficou em -0,07%, no problema rec01; e
f) Os resultados do rAG tendem a serem melhores que o GASA quando o número de tarefas e máquinas é grande.

Estes resultados mostram claramente que o rAG é ao mesmo tempo mais eficaz e eficiente que o GASA, é mais eficaz porque obtém os melhores resultados e mais eficiente porque faz isso em menos tempo.

Tabela 2 – Comparação do rAG com o GASA.

Instância	n x m	GASA	t_{GASA} (s)	rAG	t_{rAG} (s)	$(t_{rAG}/t_{GASA}) \times 2$	Desvio (%)
rec01	20x5	1527	6,00	1526	0,10	0,033	-0,07
rec03	20x5	1392	6,00	1361	0,10	0,033	-2,23
rec05	20x5	1524	7,00	1514	0,10	0,029	-0,66
rec07	20x10	2046	12,00	2043	0,10	0,017	-0,15
rec09	20x10	2045	11,00	2042	0,10	0,018	-0,15
rec11	20x10	1881	10,00	1881	0,10	0,020	0,00
hel2	20x10	180	10,00	179	0,10	0,020	-0,56
rec13	20x15	2556	17,00	2545	0,15	0,018	-0,43
rec15	20x15	2529	17,00	2529	0,15	0,018	0,00
rec17	20x15	2590	16,00	2588	0,15	0,019	-0,08
rec19	30x10	2985	34,00	2850	0,20	0,012	-4,52
rec21	30x10	2948	35,00	2827	0,20	0,011	-4,10
rec23	30x10	2827	35,00	2703	0,20	0,011	-4,39
rec25	30x15	3732	55,00	3593	0,25	0,009	-3,72
rec27	30x15	3560	51,00	3431	0,25	0,010	-3,62
rec29	30x15	3440	54,00	3303	0,25	0,009	-3,98
rec31	50x10	4757	147,00	4343	0,55	0,007	-8,70
rec33	50x10	4998	145,00	4510	0,55	0,008	-9,76
rec35	50x10	4891	146,00	4420	0,55	0,008	-9,63
rec37	75x20	9508	907,00	8203	1,30	0,003	-13,73
rec39	75x20	9964	890,00	8554	1,30	0,003	-14,15
rec41	75x20	9978	904,00	8647	1,30	0,003	-13,34
hel1	100x10	877	1088,00	730	1,95	0,004	-16,76
Média			**200,13**		**0,43**	**0,014**	**-4,99**

Analisando os dados da Tabela 3, dada a seguir, se descreve as seguintes observações:
a) Nos 23 problemas testados o rAG obteve melhor desempenho que o TS-M em 7 problemas, enquanto em 6 o desempenho foi idêntico e em 10 problemas o TS-M foi melhor. Na média o rAG foi 0,22% inferior ao desempenho do TS-M;
b) O tempo de execução do rAG para os 23 problemas foi comparativamente o mesmo usado pelo TS-M;
c) O melhor resultado do rAG, em comparação com o TS-M, foi -0,52% no problema rec19;
d) O pior resultado do rAG em comparação com o TS-M foi 1,96%, no problema hel1.

Estes resultados mostram que o rAG é inferior ao TS-M quando as condições de tempo de execução são equivalentes. Porém a diferença é muito pequena de 0,22% e lembrando que o TS-M começa de uma solução boa.

Tabela 3 – Comparação do rAG com o TS-M.

Instância	n x m	TS-M	t_{TS-M} (s)	rAG	t_{rAG} (s)	(t_{rAG}/ t_{TS-M}) x 2	Desvio (%)
rec01	20x5	1527	0,20	1526	0,10	1,00	-0,07
rec03	20x5	1361	0,20	1361	0,10	1,00	0,00
rec05	20x5	1512	0,20	1514	0,10	1,00	0,13
rec07	20x10	2042	0,20	2043	0,10	1,00	0,05
rec09	20x10	2043	0,20	2042	0,10	1,00	-0,05
rec11	20x10	1881	0,20	1881	0,10	1,00	0,00
hel2	20x10	179	0,20	179	0,10	1,00	0,00
rec13	20x15	2545	0,30	2545	0,15	1,00	0,00
rec15	20x15	2529	0,30	2529	0,15	1,00	0,00
rec17	20x15	2587	0,30	2588	0,15	1,00	0,04
rec19	30x10	2865	0,40	2850	0,20	1,00	-0,52
rec21	30x10	2825	0,40	2827	0,20	1,00	0,07
rec23	30x10	2705	0,40	2703	0,20	1,00	-0,07
rec25	30x15	3593	0,50	3593	0,25	1,00	0,00
rec27	30x15	3432	0,50	3431	0,25	1,00	-0,03
rec29	30x15	3291	0,50	3303	0,25	1,00	0,36
rec31	50x10	4347	1,10	4343	0,55	1,00	-0,09
rec33	50x10	4469	1,10	4510	0,55	1,00	0,92
rec35	50x10	4427	1,10	4420	0,55	1,00	-0,16
rec37	75x20	8127	2,60	8203	1,30	1,00	0,94
rec39	75x20	8518	2,60	8554	1,30	1,00	0,42
rec41	75x20	8543	2,60	8647	1,30	1,00	1,22
hel1	100x10	716	3,90	730	1,95	1,00	1,96
Média			**0,87**		**0,43**	**1,00**	**0,22**

4. Conclusão

O rAG mostrou que um AG que utiliza os seus princípios originais, diversificação e intensificação, de forma eficiente consegue obter bons resultados. Isto ficou comprovado com a implementação dos três procedimentos inspirados nesses princípios, que no primeiro experimento diminuiu o desvio das soluções do rAG de 1,710% para 0,171%, uma melhoria de 10 vezes. Este resultado aponta a importância de um bom projeto para os componentes originais do AG, antes de recorrer a inicialização eficiente e hibridização para tornar o AG competitivo em relação a outros métodos de otimização.

O rAG mostrou ser o melhor AG para o CPFSP sendo o *makespan* a função objetivo (F/no-wait/C_{max}), porque foi 4,99% melhor que o AG de Shuster e Framinan (2003). O rAG foi 0,22% inferior ao TS-M de Grabowski e Pempera (2005), o melhor método para esse problema, nas mesmas condições de tempo de execução. Porém, o rAG se torna melhor quando usa 4,46 vezes mais tempo de execução, obtendo um desvio médio 0,16% melhor que o TS-M.

O rAG mostrou quando um AG é bem projetado se torna competitivo ou até melhor que métodos baseados em metaheurísticas que trabalham sobre uma solução de cada vez, como *Tabu Search* e *Simulated Anealing*. Ainda, o rAG mostrou que um AG consegue obter melhores soluções que outros métodos sem precisar recorrer a inicialização eficiente ou hibridização, mostrando que se os princípios do AG forem bem trabalhados esse método consegue encontrar bons resultados.

Propostas para futuros trabalhos:

- Incorporar o atributo reativo à mutação populacional. Pois a mutação populacional realiza apenas uma perturbação e sempre no mesmo intervalo de gerações sem melhoria. Uma sugestão é poder realizar mais de uma perturbação de cada vez e em intervalos de geração diferentes, dependendo da quantidade de tarefas e máquinas do problema;
- Como o rAG obteve bons resultados para o CPFSP mesmo sem inicialização eficiente, uma proposta para melhorar o rAG seria implementar uma inicialização eficiente que não comprometa a diversidade da população;
- Poderia ser testados outros valores para os parâmetros do rAG; e
- Aplicar o rAG em outros problemas POCP e comparar o seu resultado com os melhores métodos desses problemas.

Agradecimentos
Os autores agradecem o apoio da UFC, CAPES e CNPq (processo 311682/2006-5).

5. Bibliografia

Aldowaisan, T., Allahverdi, A., New heuristics for no-wait flowshops to minimize makespan. *Computers and Operations Research*, 30, 1219-1231, 2003.
Chen, C.L., Vempati, V.S., Aljaber, N., An application of genetic algorithms for flow shop problems. *European Journal of Operational Research* 80, 389–396, 1995.
Chen, C.L., Neppalli, R.V., Aljaber, N., Genetic algorithms applied to the continuous flow shop problem. *Computers and Industrial Engineering*, 30, 919-929, 1996.
DeJong, K.A., Adaptive systems design: a genetic approach. *IEEE Trans. Syst., Man, Cyber. SMC-16*, 566-574, 1980.
Dréo, J., **Pétrowski**, A., **Siarry**, P., **Taillard, E.**, *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer, New York, 2006.
Fink, A., Voβ, S., Solving the continuous flow-shop scheduling problem by metaheuristics. *European Journal of Operational Research*, 151, 400-414, 2003.
Gomes, F.R.A., Silva, J.L.C., O problema de sequenciamento *flowshop*: uma abordagem evolucionária. *Anais do XXXIX SBPO - Simpósio Brasileiro de Pesquisa Operacional*, Fortaleza-CE, 2007.
Grabowski, J., Pempera, J., Some local search algorithms for no-wait flow-shop problem with makespan criterion. *Computers and Operations Research*, 32, 2197-2212, 2005.
Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R., Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
Grefenstette, J.J., Optimized control of parameters for genetic algorithms. *IEEE Trans. Syst., Man, Cyber. SMC-16*, 122-128, 1986.
Hall, N.G., Sriskandarajah, C., A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research* 44, 510-525, 1994.
Haupt, R.L, Haupt, S.E., *Pratical Genetic Algorithms*. John Wiley & Sons Inc., Hoboken, 2004.
Heller, J., Some experiments for an M x J flow shop and its decision-theoretical aspects. *Operations Research*, 8, 178-184, 1960.
Holland, J.H., *Adaptation in Natural Artificial Systems*. University of Michigan Press, Michigan, 1975.
Lobo, F.G.; Goldberg, D.E., The parameter-less genetic algorithm in practice. *Information Sciences*, 167, 217-232, 2004.
Mitchell, M., *An introduction to Genetic Algorithms*. MIT Press, Cambridge, 1998.
Nawaz, M., Enscore, E., Ham, I., A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *The International Journal of Management Sciences*, 11, 91-95, 1983.
Reeves, C.R., A genetic algorithm for flow shop sequencing. *Computers and Operations Research*, 22, 5-13, 1995.

Reeves, C.R., Rowe, J.E, *Genetic Algorithms: Principles and Perspectives: a Guide to GA Theory*. Kluwer Academic Publishers, New York, 2002.
Röck, H., The three-machine no-wait flowshop problem is NP-complete. *Journal of the Association for Computing Machinery*, 31, 336-345, 1984.
Ruiz, R., Maroto, C., Alcaraz, J., Two newrobust genetic algorithms for the flowshop scheduling problem. *The International Journal the Management Science*, 34, 461-476, 2006.
Schuster, C.J., Framinan, J.M., Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, 31, 308-318, 2003.
Silva, J.L.C., Soma, N.Y., Uma heurística para problemas de otimização combinatória permutacional. *Anais do XXXIII SBPO - Simpósio Brasileiro de Pesquisa Operacional*, Campos do Jordão-SP, 1298-1306, 2001.
Silva, J.L.C., Soma, N.Y., Um método heurístico aplicado no problema de programação flow shop permutacional. *Anais do XXXVIII SBPO - Simpósio Brasileiro de Pesquisa Operacional*, Goiânia-GO, 2006a.
Silva, J.L.C., Soma, N. Y., Um Algoritmo Genético Híbrido Construtivo polinomial aplicado ao Flowshop Scheduling Problem. *Anais do XIII CLAIO-Congresso Latino-Iberoamericano de Investigación Operativa*, Montevideo-Uruguay, 2006b.