O PROBLEMA DE SEQÜENCIAMENTO FLOWSHOP: UMA ABORDAGEM EVOLUCIONÁRIA

Francisco Régis Abreu Gomes
Pós-Graduação em Logística e Pesquisa Operacional-UFC
Campus do Pici, Bloco 703, CEP 60455, Fortaleza-CE

José Lassance de Castro Silva
Universidade Federal do Ceará-UFC
Campus do Pici, Bloco 910, CEP 60000-000, Fortaleza-CE

Resumo

Neste trabalho descrevemos uma nova metodologia aplicada na resolução do problema de Seqüenciamento *flowshop* (*FSP*), através de algoritmos evolucionários. A técnica apresentada para resolver o FSP, baseada em algoritmos genéticos, também pode ser aplicada a outros Problemas de Otimização Combinatória Permutacional. Ela é simples de programar computacionalmente devido à estrutura usada na modelagem do problema. O FSP pertence à classe dos problemas NP-difícil, que justifica o uso de técnicas refinadas aplicadas na resolução do mesmo com o intuito de encontrar boas soluções viáveis. Extensivos experimentos computacionais foram realizados e reportados para instâncias do problema com até 100 tarefas e 20 máquinas, e os resultados são comparados com aqueles encontrados na literatura.
Palavras-chave: Problema de Seqüenciamento, Otimização Combinatória, Heurística.

Abstract

The aim of this paper is to present a new method applied to solve the FlowShop Scheduling Problem (*FSP*), through evolucionary algorithm. The technique presented to solve FSP, based on genetic algorithms, it can also be applied the other Combinatorial Optimization Problems and it is a simple program of computer where the solutions are based on permutation. This technique was applied effectively to the FSP which is an NP-hard problem and difficult to be solved in the practice. Extensive computational experiments are reported for instances with up to 100 jobs and 20 machines and the results are compared with those obtained from the literature.
Key words: Scheduling Problem, Combinatorial Optimization, Heuristics.

1. Introdução

O problema de programação de operações em um ambiente *Flow Shop* é um problema de programação de produção no qual *n* tarefas devem ser processadas por um conjunto de *m* máquinas distintas, tendo o mesmo fluxo de processamento nas máquinas. Uma nova denominação é dada ao problema quando as tarefas possuem fluxos distintos nas máquinas: *Job Shop*. Usualmente a solução do problema consiste em determinar uma seqüência das tarefas dentre as (*n*!) seqüências possíveis, que é mantida para todas as máquinas (programação permutacional) e que geralmente procura minimizar a duração total da programação (*makespan*), ou seja, o intervalo de tempo entre o início de execução da primeira tarefa na primeira máquina e o término de execução da última tarefa na última máquina.

O Problema de Seqüenciamento *Flowshop*, denominado na literatura de *Flowshop Scheduling Problem* (FSP), possuí as seguintes características:

a) É dado um conjunto com *n* tarefas $J_1, J_2,..., J_n$;

b) É dado um conjunto com *m* máquinas $M_1, M_2, ..., M_m$;

c) Cada tarefa demanda *m* operações, com uma operação representando o tempo de processamento da tarefa por máquina;
d) As tarefas seguem o mesmo fluxo de operações nas máquinas, isto 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) Caso a tarefa Jj não utilize todas as máquinas, o seu fluxo continua sendo o mesmo, todavia com o tempo de ocupação sendo igual a zero;
f) Uma máquina pode processar somente uma operação de cada vez, e iniciada uma operação, ela deva ser processada até a sua conclusão;
g) O número de seqüências distintas possíveis para realização das tarefas nas máquinas é grande, i. e. *O*(n!).

Um *input* FSP é dado por *n*, *m* e uma matriz *P*(*n* x *m*) de elementos não negativos, onde P_{ij} denota o tempo de processamento da tarefa J_j na máquina M_i. Seguindo os 4 parâmetros da notação *A*/*B*/*C*/*D* adotada por Conway *et al.* (1967), o problema é classificado como $n/m/P/F_{max}$. Na recente notação paramétrica α/β/γ, proposta por Graham *et al.* (1979), o problema é denotado como sendo F/prmu/C_{max}. O FSP pertence à classe dos problemas NP-*hard*, quando $m \geq 3$, conforme Garey *et al*. (1976), de forma que pode ser resolvido eficientemente de maneira ótima somente em casos de pequeno porte. No caso em que $m=2$, o problema pode ser solucionado através de um algoritmo em tempo polinomial, Johnson (1954).

Gupta e Stafford Jr. (2006), fizeram um levantamento científico sobre o problema nas últimas cinco décadas e, constataram que existem mais de 1200 artigos, na literatura sobre pesquisa operacional, contendo vários aspectos deste problema. Os métodos de resolução exata, geralmente, são aplicados a problemas de pequena instância ($n \leq 20$), e mesmo neste caso o tempo computacional ainda é muito alto. Ruiz *et al.* (2006) aborda, de forma resumida, alguns métodos utilizados na resolução do problema de forma aproximativa, através de heurísticas e meta-heurísticas. Dentre elas, podemos citar as heurísticas: Palmer (1965); Campbell *et al.* (1970); Gupta (1971); Nawaz *et al.*, (1983); Hundal e Rajgopal (1988); Moccellin (1999); e as meta-heurísticas: AGChen, Chen *et al.* (1995); e AGHC, de Silva e Soma (2006). Técnicas de Programação Matemática, tais como Programação Linear Inteira, Selen e Hott (1986), Wilson (1989), e técnicas de enumeração do tipo *branch-and-bound*, Ignal e Schrage (1965), Potts (1980), têm sido empregadas para a solução ótima do problema. Entretanto, tais técnicas não são eficientes em termos computacionais, em problemas de médio e grande porte. Desta forma, justifica-se a utilização de métodos heurísticos usados na resolução do problema.

Conforme Reeves (1995), as heurísticas foram desenvolvidas com a finalidade de resolver problemas de elevado nível de complexidade em tempo computacional razoável. Ao se pensar em um problema combinatório complexo, uma opção seria analisar todas as combinações possíveis para conhecer a melhor delas. Whitley *et al.* (1991) fizeram uma boa abordagem da relação do problema de seqüenciamento com o problema do caixeiro viajante, onde constataram que as técnicas utilizadas na resolução de um dos problemas também podem ser aplicadas na resolução do outro, com pequenas modificações.

Um Problema de Otimização Combinatória Permutacional (*POCP*) pode ser definido por um terno (*S*, *g*, *n*), onde *S* é o conjunto de todas as soluções viáveis (soluções que satisfazem as restrições do problema, com $| S | = n!$), *g* é a função objetiva que aplica a cada solução $s \in S$ um número real e *n* é uma instância do problema. O objetivo é encontrar a solução $s \in S$ que minimize a função objetiva *g*. Podemos representar *s* como uma permutação de *n* elementos distintos, ou seja, $s = < a_1\ a_2 \ldots a_n >$. N(*s*) é chamada *a vizinhança de s* e contém todas as soluções que podem ser alcançadas de *s* por um simples movimento. Aqui, o significado de um *movimento* é aquele de um operador que transforma uma solução para uma outra com pequenas modificações.

O FSP pode ser modelado como um POCP (*S*, *g*, *n*), tendo a 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;
b) O procedimento (h) adotado para avaliar uma solução do problema é dado a seguir, onde ele determina o valor do tempo gasto (tg) para processar a seqüência s, mais precisamente tem-se que tg é o tempo de finalização do processamento da última tarefa de s na máquina M_m.

Entrada: m, n, permutação s, Matrizes $T(m \times n)^*$ e $P(m \times n)$.
Saída: tg (tempo gasto para processar todas as n tarefas usando a seqüência s)

```
for(i=1; i<=m; i++)
        for(j=1; j<=n; j++)  t[i][j]=0;
for(j=1; j<=n; j++) {
        for(i=1; i<=m; i++) {
                if (i==1) {
                        if (j>=2) t[1][s[j]]=t[1][s[j-1]]+p[1][s[j-1]];
                } else {
                        if (j==1)
                                {t[i][s[1]]=t[i-1][s[1]]+p[i-1][s[1]];
                        } else {
                                x=t[i][s[j-1]]+p[i][s[j-1]];
                                y=t[i-1][s[j-1]]+p[i-1][s[j-1]];
                                if (x>=y) t[i][s[j]]=x; else t[i][s[j]]=y;
                        }
                }
        }
}
tg=t[m][s[n]] + p[m][s[n]];
```

* O elemento t_{ij} representa o tempo para iniciar a tarefa J_j na máquina M_i.

O ataque ao FSP sugerido aqui se baseia na teoria dos Algoritmos Genéticos (*AG*), da computação evolucionária, desenvolvido por Holland (1975). Resolvemos implementar um algoritmo genético que tenta diversificar e intensificar a busca por boas soluções dentro dos AGs tradicionais. Usamos os algoritmos genéticos propostos por Silva e Soma (2006), denominado *AGHC*, e *AGChen* desenvolvido por Chen *et al*. (1995), para comparar o desempenho da nossa técnica, dentro desta metodologia. No AGChen, a aptidão de cada indivíduo é igual a diferença entre o maior *makespan* de um indivíduo da população atual e o *makespan* do indivíduo que se quer calcular. O tamanho da população é de 60 indivíduos, a população inicial é criada a partir dos métodos CDS e Danninbring e o algoritmo é finalizado depois de 200 gerações. O método de seleção adotado é o da roleta. A taxa de *crossover* adotada foi de 100% o que não é usual, pois não preserva nenhum indivíduo para a próxima geração. O operador de mutação não foi implementado. Chen *et al*. afirmam que essas modificações melhoraram a eficiência do AG em seus testes. Ruiz *et al.* (2006) fornecem os resultados médios do desvio obtido pelo AGChen para os problemas propostos na OR-library (Beasley, 1990), instâncias de Taillard (1993). No AGHC a população inicial é criada a partir do método HP, em cada iteração o algoritmo utiliza uma população diferente formada por 52 indivíduos, divididos em 4 grupos de 13 indivíduos cada. As populações são geradas e controladas por um procedimento que evita repetição de indivíduos nas mesmas. O cruzamento é feito com todos os indivíduos do grupo 1 com os do grupo 2, da mesma forma os do grupo 3 com os do grupo 4. Para realizar o procedimento de mutação é utilizada a heurística 3-Opt de Lin e Kernighan.

Segundo Simões (1999) grande parte dos AG desenvolvidos para a resolução de problemas específicos, afasta-se das idéias básicas da genética, utilizando operadores que dependem do domínio do problema abordado. Sem criticar esses AG, alguns autores alertam para a necessidade de que os

modelos computacionais sejam mais próximos dos modelos biológicos (Simões, 1999). Tanto o AGChen como o AGHC seguem essa tendência, por isso o objetivo desse trabalho é apresentar um AG puro, denominado de *rAG*, que não foge muito do que foi proposto originalmente por Holland (1975).

A organização do artigo segue à: Na Seção 2, apresentaremos as principais idéias de rAG. Na Seção 3, serão apresentados os experimentos computacionais, com os resultados obtidos. Finalizamos nosso trabalho com a apresentação das conclusões sobre a nossa pesquisa, Seção 4, enquanto na Seção 5 tem-se o material bibliográfico consultado.

2. O Algoritmo Genético (rAG)

O rAG, dado abaixo, é uma técnica de otimização baseada nos princípios da genética e da seleção natural. Esta técnica utiliza uma população de indivíduos que evoluem baseados em regras específicas, com o objetivo de maximizar a aptidão da população. A implementação do rAG se fez em quatro etapas: 2.1) Codificação dos cromossomos; 2.2) Definição da função de aptidão; 2.3) Tamanho e geração da população; 2.4) Aplicação dos operadores genéticos (crossover, mutação e desconvergência); e 2.5) Critério de parada. A seguir descrevemos cada uma dessas etapas.

Algoritmo rAG
Geração e Avaliação da população inicial
Enquanto critério de parada for falso faça
    Seleção
    Crossover
    Mutação
    Desconvergência

2.1. Codificação do Cromossomo

O *cromossomo* (indivíduo da população) é definido como sendo uma permutação das *n* tarefas, onde cada tarefa representa um gene do cromossomo. A ordem na qual as tarefas se encontram no cromossomo determina a ordem de processamento das mesmas nas máquinas.

2.2. Definição da Função de Aptidão

Usamos o procedimento *h*, descrito anteriormente, que é igual ao *makespan* do problema para ser a função de aptidão (*fitness*) de cada indivíduo.

2.3. Tamanho e geração da população inicial

Consideramos o tamanho da população de 75 indivíduos (cromossomos), gerada da seguinte forma:

1. Geramos um cromossomo (permutação) aleatoriamente;
2. Os demais 74 cromossomos são gerados, um a um, selecionando aleatoriamente 2 genes distintos do primeiro cromossomo, entre 1 e n, e trocando de posição. A Figura 1, abaixo, mostra um exemplo deste procedimento.

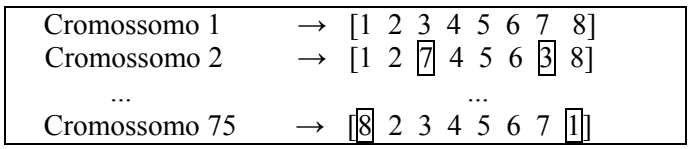

Figura 1 – Ilustração da população inicial para um problema com *n*=8.

2.4. Aplicação dos Operadores Genéticos

A seleção é o primeiro operador genético a ser aplicado, que também depois de alguns experimentos se constatou que a melhor estratégia era a seleção por torneio com elitismo igual a dois, dentre várias opções avaliadas. A seleção por torneio consiste em sortear dois ou mais cromossomos

aleatoriamente e, aquele com maior aptidão ser selecionado, já o elitismo consiste em garantir que o melhor ou os melhores indivíduos serão mantidos para a próxima população (Haupt e Haupt, 2004).

O segundo operador genético é o crossover, onde se optou pelo crossover uniforme, que é uma generalização do crossover de um ponto. Nesse caso, o número de pontos é igual ao número de gens, os pontos são representados por uma cadeia de 0's e 1's, denominada de máscara (Haupt e Haupt, 2004). O procedimento abaixo descreve o operador crossover, com a Figura 2, descrevendo um exemplo desta aplicação para n=8.

1º) Gera-se uma máscara de forma aleatória;
2º) O descendente recebe o gen do *Pai 1* quando o valor da máscara, correspondente, é 0 e, recebe o valor do gen do *Pai 2* quando o valor da máscara é 1;
3º) Quando houver um gen repetido, ele é substituído pelo índice da menor tarefa não presente no cromossomo repetido de menor índice. Este procedimento finaliza-se quando não há gen repetido.

Pai 1	→	[2 3 1 4 5 6 8 7]
Pai 2	→	[2 6 7 4 5 3 8 1]
Máscara	→	[0 1 1 0 1 0 1 0]
Resultado	→	[2 6 7 4 5 6 8 7]
Cromossomo	→	[2 6 7 4 5 1 8 3]

Figura 2 – Aplicação do operador crossover.

O terceiro operador é a mutação, que pode inserir novos indivíduos à população explorando novas áreas do espaço de busca (Haupt e Haupt, 2004). O procedimento do operador de mutação consiste de trocar dois genes de posição selecionados aleatoriamente.

A geração da população inicial consistia em aplicar o operador de mutação, não provocando uma maior diversificação, o algoritmo rapidamente convergia para um ótimo local, por isso, desenvolveu-se um novo procedimento chamado de *desconvergência*. Este procedimento foi desenvolvido na tentativa de evitar uma convergência prematura do método e diversificar a busca, com base na seguinte informação: se dois ou mais cromossomos tiverem o mesmo *makespan*, um permanece inalterado, enquanto os outros sofrerão mutação. O cromossomo que permanece inalterado é o de menor "ordem", por exemplo, se os cromossomos 2, 60 e 65 possuem o mesmo *makespan* o cromossomo 2 permanece inalterado e os cromossomos 60 e 65 sofrem mutação, podendo ter um novo *makespan*.

2.5 Critério de Parada

O critério utilizado para finalizar a execução do rAG foi o número de iterações. No final da seção 3.1 é definido o número de iterações para o rAG.

3. Experimentos Computacionais

Vários experimentos computacionais foram realizados para definir os valores parametrizados e o desempenho do método que estão descritos nas seções 3.1 e 3.2, respectivamente.

3.1. Definição dos Parâmetros de rAG

Foram realizados testes com diferentes valores para a taxa de crossover, tamanho da população e número de iterações a serem utilizados em rAG. Fizemos um estudo combinando os valores atribuídos para a taxa de crossover (70% e 75%) e o tamanho da população (50, 75 e 100), embora outros valores foram utilizados mas que não estão apresentados porque tiveram um comportamento muito similar aos que aqui se encontram. A Tabela 1 e a Figura 3, dadas abaixo, mostram as combinações configuradas e avaliadas.

Combinação	Taxa de crossover	População	Combinação	Taxa de crossover	População

1	70%	50	4	75%	75
2	75%	50	5	70%	100
3	70%	75	6	75%	100

Tabela 1 – Combinação dos valores para taxa de crossover e tamanho da população.

Na Figura 3, dada abaixo, é apresentado o gráfico da avaliação das combinações 1, 2, 3, 4, 5 e 6, no desempenho dos 10 problemas da primeira classe (n=50 e m=20). Através deste gráfico é possível ver que a combinação 3, foi aquela que teve em média o melhor resultado. Com isso adotamos a taxa de *crossover* e o tamanho da população de 70% e 75, respectivamente.

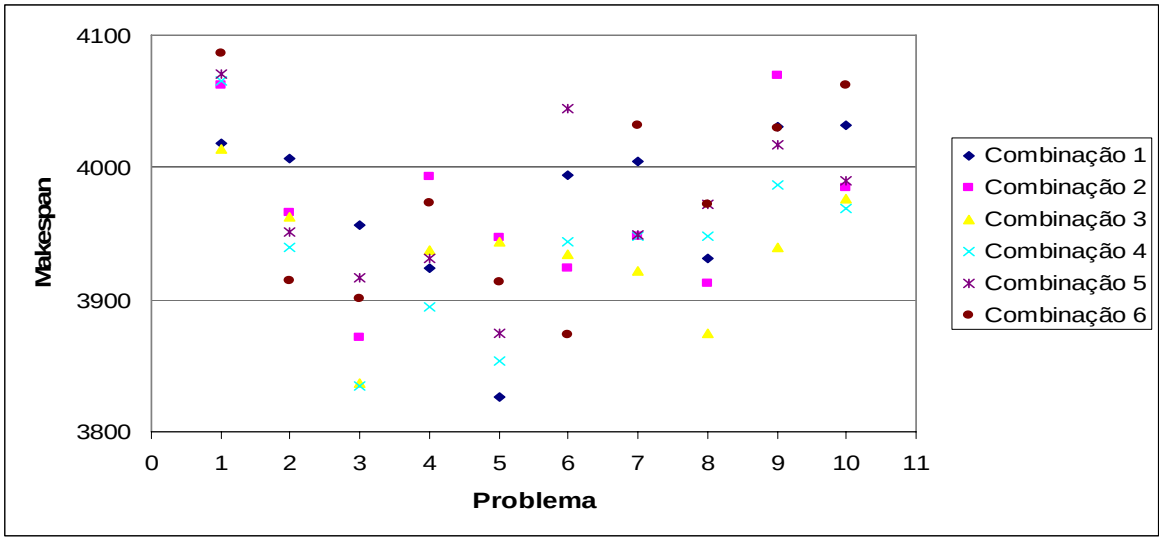

Figura 3 – Desempenho dos testes realizados para definir valores iniciais do rAG.

Uma análise também foi feita para o Número de Iterações (*NI*), onde pudemos constatar que para problemas com $n \leq 50$ não houve grande mudança na qualidade das soluções apresentadas, quando *NI* foi superior a 350 e, para o valor de n=100 o *NI* ideal teve uma variação de 800 a 900, sem que comprometesse o tempo computacional de execução do método.

Os tempos de execução para todos os problemas testados foram quase que semelhantes, nas devidas proporções dos valores de n. Os parâmetro de rAG ficaram assim definidos: Tamanho da população: 75; Elitismo: 2; Taxa de *crossover*: 70 %; Taxa de mutação: 5 %; Número do torneio: 3; e *NI*: 350, se $n \leq 50$, e 900, se $n > 50$.

3.2. Resultados

Vários experimentos computacionais foram realizados para observamos o desempenho de rAG, executado em um PC-AMD (2.2 GHz, 256 Mb de RAM), tendo sido o código implementado em linguagem delphi 7.0. A Tabela 2, a seguir, mostra o desempenho do método quando aplicado nas instâncias de 1 a 90, *benchmarks* de Taillard (1993), encontradas na OR-Library, em http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/flowshop2.txt. Nesta tabela encontram-se: as instâncias baseadas nos valores de m e n; os desvios ($100 \times (z - z^*)/z^*$, onde z é o valor de h na melhor solução encontrada e z^* é o valor ótimo) de rAG e AGHC; o tempo gasto, em segundos, para executar cada um dos inputs da classe para AGHC e rAG; e a diferença entre os desvio de AGHC e rAG.

Instância	Z*	AGHC			rAG			AGHC - rAG (%)
		Z	Tempo (s)	Desvio (%)	Z	Tempo (s)	Desvio (%)	
20x5								
1	1278	1326	0,00	3,76	1297	0,234	1,49	2,27
2	1359	1383	1,00	1,77	1377	0,234	1,32	0,44
3	1081	1185	0,00	9,62	1093	0,235	1,11	8,51

4	1293	1388	0,00	7,35	1331	0,234	2,94	4,41
5	1235	1302	0,00	5,43	1250	0,234	1,21	4,21
6	1195	1272	0,00	6,44	1210	0,25	1,26	5,19
7	1239	1304	1,00	5,25	1251	0,234	0,97	4,28
8	1206	1309	0,00	8,54	1214	0,235	0,66	7,88
9	1230	1348	0,00	9,59	1273	0,234	3,50	6,10
10	1108	1212	0,00	9,39	1131	0,25	2,08	7,31
20x10								
11	1582	1740	1,00	9,99	1628	0,312	2,91	7,08
12	1659	1814	0,00	9,34	1726	0,312	4,04	5,30
13	1496	1669	1,00	11,56	1549	0,328	3,54	8,02
14	1377	1570	0,00	14,02	1435	0,313	4,21	9,80
15	1419	1594	0,00	12,33	1461	0,312	2,96	9,37
16	1397	1544	1,00	10,52	1446	0,312	3,51	7,02
17	1484	1633	0,00	10,04	1508	0,313	1,62	8,42
18	1538	1726	0,00	12,22	1559	0,328	1,37	10,86
19	1593	1689	0,00	6,03	1636	0,313	2,70	3,33
20	1591	1755	0,00	10,31	1629	0,328	2,39	7,92
20x20								
21	2297	2494	1,00	8,58	2363	0,672	2,87	5,70
22	2099	2298	1,00	9,48	2130	0,656	1,48	8,00
23	2326	2502	0,00	7,57	2379	0,656	2,28	5,29
24	2223	2454	1,00	10,39	2275	0,671	2,34	8,05
25	2291	2502	1,00	9,21	2456	0,672	7,20	2,01
26	2226	2401	1,00	7,86	2269	0,672	1,93	5,93
27	2273	2443	0,00	7,48	2349	0,671	3,34	4,14
28	2200	2412	1,00	9,64	2260	0,672	2,73	6,91
29	2237	2438	1,00	8,99	2302	0,656	2,91	6,08
30	2178	2401	0,00	10,24	2274	0,672	4,41	5,83
50x5								
31	2724	2835	2,00	4,07	2752	1,297	1,03	3,05
32	2834	2942	1,00	3,81	2890	1,281	1,98	1,83
33	2621	2719	2,00	3,74	2648	1,265	1,03	2,71
34	2751	2931	1,00	6,54	2777	1,282	0,95	5,60
35	2863	2983	1,00	4,19	2864	1,297	0,03	4,16
36	2829	2960	2,00	4,63	2835	1,281	0,21	4,42
37	2725	2855	1,00	4,77	2746	1,297	0,77	4,00
38	2683	2849	1,00	6,19	2722	1,281	1,45	4,73
39	2552	2706	2,00	6,03	2578	1,296	1,02	5,02
40	2782	2890	1,00	3,88	2789	1,25	0,25	3,63
50x10								
41	2991	3453	2,00	15,45	3140	2,062	4,98	10,46
42	2867	3328	2,00	16,08	3010	2,109	4,99	11,09
43	2839	3255	2,00	14,65	2969	2,078	4,58	10,07
44	3063	3456	2,00	12,83	3126	2,078	2,06	10,77
45	2976	3445	2,00	15,76	3120	2,063	4,84	10,92
46	3006	3412	2,00	13,51	3159	2,062	5,09	8,42
47	3093	3481	2,00	12,54	3289	2,094	6,34	6,21
48	3037	3403	3,00	12,05	3128	2,094	3,00	9,05
49	2897	3325	2,00	14,77	3032	2,094	4,66	10,11
50	3065	3512	2,00	14,58	3204	2,109	4,54	10,05

50x20								
51	3771	4485	3,00	18,93	4054	3,719	7,50	11,43
52	3668	4259	4,00	16,11	3926	3,703	7,03	9,08
53	3591	4270	4,00	18,91	3865	3,687	7,63	11,28
54	3635	4353	4,00	19,75	3951	3,718	8,69	11,06
55	3553	4294	4,00	20,86	3865	3,703	8,78	12,07
56	3667	4273	4,00	16,53	3947	3,687	7,64	8,89
57	3672	4354	4,00	18,57	3908	3,75	6,43	12,15
58	3627	4311	4,00	18,86	3932	3,672	8,41	10,45
59	3645	4356	4,00	19,51	3972	3,719	8,97	10,53
60	3696	4415	4,00	19,45	4003	3,718	8,31	11,15
100x5								
61	5493	5673	7	3,28	5527	6,297	0,62	2,66
62	5268	5485	7	4,12	5290	6,281	0,42	3,70
63	5175	5392	6	4,19	5213	6,219	0,73	3,46
64	5014	5126	7	2,23	5035	6,203	0,42	1,81
65	5250	5460	7	4,00	5311	6,172	1,16	2,84
66	5135	5332	6	3,84	5146	6,297	0,21	3,62
67	5246	5439	7	3,68	5284	6,328	0,72	2,95
68	5034	5284	6	4,97	5137	6,218	2,05	2,92
69	5448	5662	6	3,93	5467	6,297	0,35	3,58
70	5322	5504	6	3,42	5343	6,297	0,39	3,03
100x10								
71	5770	6354	10	10,12	5943	9,703	3,00	7,12
72	5349	6012	10	12,39	5449	9,641	1,87	10,53
73	5676	6142	10	8,21	5807	9,844	2,31	5,90
74	5781	6439	10	11,38	6005	9,656	3,87	7,51
75	5467	6158	10	12,64	5627	9,672	2,93	9,71
76	5303	5902	10	11,30	5438	9,734	2,55	8,75
77	5595	6156	10	10,03	5758	9,719	2,91	7,11
78	5617	6189	10	10,18	5729	9,781	1,99	8,19
79	5871	6428	10	9,49	5979	9,813	1,84	7,65
80	5845	6337	10	8,42	5939	9,782	1,61	6,81
100x20								
81	6106	7292	17	19,42	6529	16,421	6,93	12,50
82	6183	7253	16	17,31	6497	16,328	5,08	12,23
83	6252	7282	16	16,47	6696	16,468	7,10	9,37
84	6254	7169	19	14,63	6681	16,25	6,83	7,80
85	6262	7342	16	17,25	6651	16,469	6,21	11,03
86	6302	7315	17	16,07	6688	16,375	6,13	9,95
87	6184	7374	17	19,24	6710	16,622	8,51	10,74
88	6315	7503	19	18,81	6811	16,531	7,85	10,96
89	6204	7350	16	18,47	6635	16,359	6,95	11,52
90	6404	7449	17	16,32	6768	16,625	5,68	10,63

Tabela 2 – Comparação dos resultados entre AGHC e rAG para n=20, 50, 100, e m=5, 10, 20.

As Figuras 4 e 5 apresentam mais detalhadamente os resultados obtidos com os experimentos computacionais descritos na Tabela 2, mostrando mais especificamente o comportamento dos métodos rAG e AGHC.

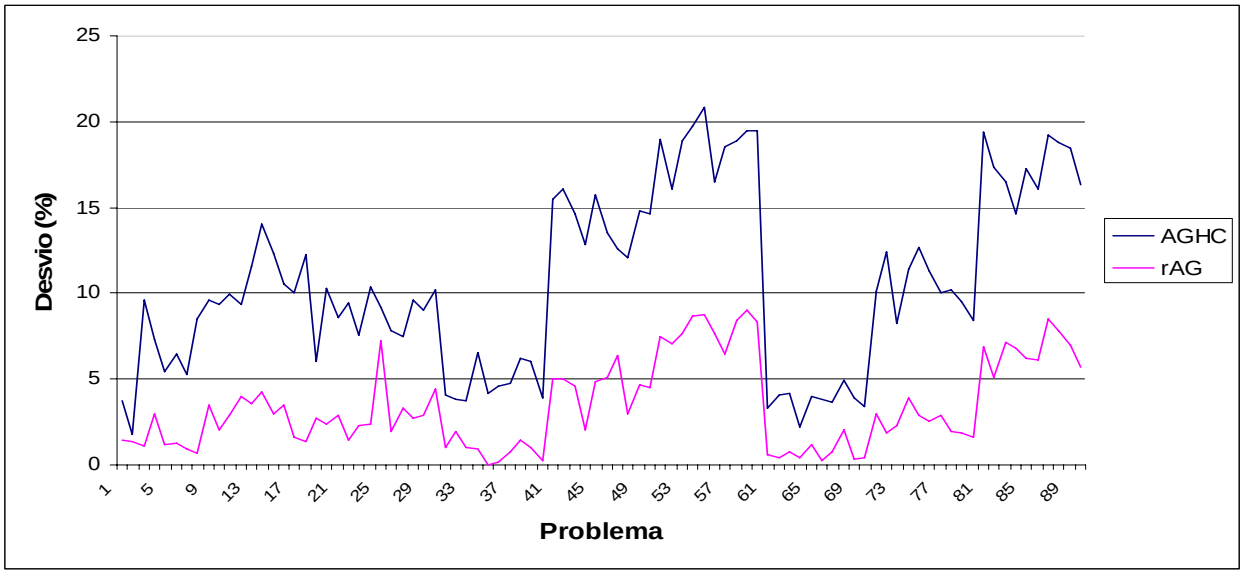

Figura 4 – Resultados de AGHC e rAG em relação ao desvio.

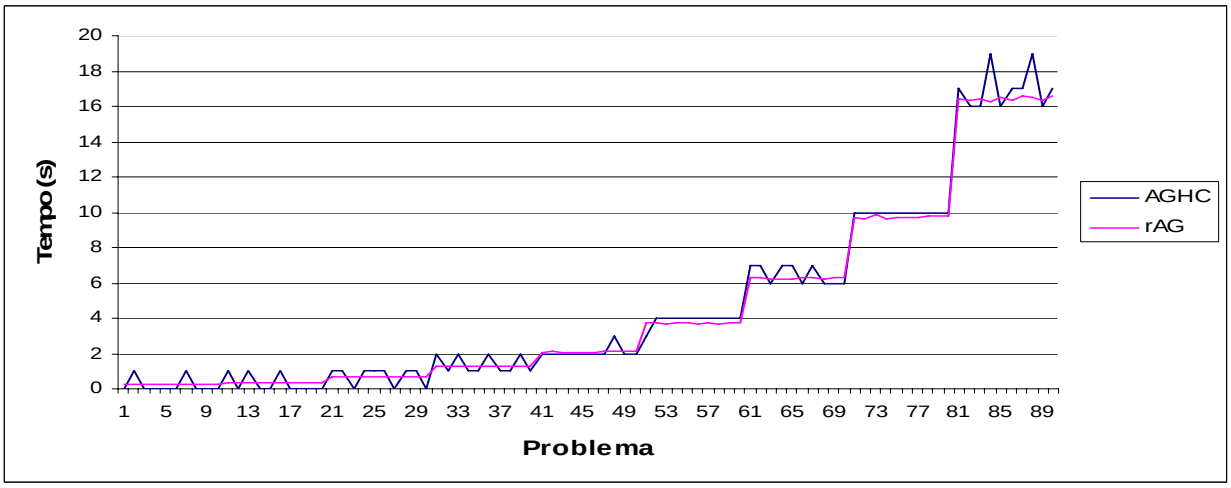

Figura 5 – Resultados de AGHC e rAG em relação ao tempo de processamento.

A Tabela 3, dada a seguir, mostra a média do desvio e do tempo de processamento para os problemas por classe, assim também como o mínimo, a média e o máximo global. Na literatura, encontramos os resultados do AGChen por classe (Chen *et al.*, 1995) e não por Instância.

A seguir descrevemos alguns resultados importantes sobre os experimentos computacionais realizados:

1. O rAG foi o método que mais se aproximou do valor ótimo, ele alcançou na instância de número 35 o melhor desvio (0,03 %), de todas as instâncias executadas;
2. O pior desempenho do rAG foi alcançado na instância de número 59, que pertence a classe de problemas 50x20, com um desvio de 8,97 %. Neste mesmo problema AGHC alcançou um desvio de 19,51 %;
3. O resultado do rAG mais próximo do AGHC foi alcançado na instância de número 2, com uma diferença entre os desvios de 0,44%;
4. O melhor desempenho do rAG sobre o AGHC foi obtido na instância de número 81, que pertence ao grupo de problemas 100x20, com uma diferença de desvio de 12,50%;
5. O rAG só não foi melhor que o AGChen em apenas uma classe, instâncias do tipo 50x20, onde o rAG teve uma média de desvio igual a 7,94% enquanto o AGChen teve 7,88%. Vale

ressaltar que o AGChen utilizou um tempo médio de 30 segundos enquanto o rAG utilizou 3,7 s;

6. Em termos gerais, o rAG apresentou melhores soluções que o AGHC e AGChen, com o rAG tendo um desvio médio global de 3,44 %, enquanto o AGHC e AGChen obtiveram de desvio médio global de 10,63 % e 4,67 %, respectivamente;
7. Dos três métodos utilizados na resolução dos problemas, o AGHC foi o método que obteve os piores resultados alcançados, como mostra o gráfico da Figura 4;
8. Ainda sobre os desvios, por classe de problemas, o rAG obteve a menor média (0,71 %) enquanto AGChen e AGHC obtiveram a segunda (1,34 %) e terceira (3,77 %), respectivamente, conforme mostra a Figura 4 mais detalhes sobre o comportamento dos métodos por classe de problemas;
9. Quanto ao tempo de processamento, tem-se que AGHC e rAG utilizaram praticamente os mesmos tempos, 4,7 e 4,5 segundos, respectivamente, enquanto AGChen obteve 30 segundos em todas as instâncias.

Instância (*n* x *m*)	**Desvio (%)**			**Tempo (s)**		
	rAG	**AGHC**	**AGChen**	**rAG**	**AGHC**	**AGChen**
20x5	1,65	6,71	3,65	0,2	0,2	30,0
20x10	2,92	10,64	5,00	0,3	0,3	30,0
20x20	3,15	8,19	3,90	0,7	0,7	30,0
50x5	0,90	4,79	1,89	1,3	1,4	30,0
50x10	4,51	12,39	6,37	2,1	2,1	30,0
50x20	7,94	18,05	7,88	3,7	3,9	30,0
100x5	0,71	3,77	1,34	6,3	6,5	30,0
100x10	2,49	4,53	3,90	9,7	10,0	30,0
100x20	6,73	10,32	8,06	16,4	17,0	30,0
Média	**3,44**	**8,82**	**4,67**	**4,5**	**4,7**	**30,0**
Mínimo	**0,71**	**3,77**	**1,34**	**0,2**	**0,2**	**30,0**
Máximo	**7,94**	**18,05**	**8,06**	**16,4**	**17,0**	**30,0**

Tabela 3 – Síntese dos resultados obtidos com rAG, AGHC e AGChen.

4. Conclusão

Uma maneira de reduzir a complexidade na resolução do problema computacionalmente é através do uso de heurísticas, que embora não garantam a solução exata, estabelece um compromisso entre os resultados obtidos e o custo computacional. Nossos experimentos computacionais, baseados em problemas da literatura, indicam que rAG obteve sucesso na resolução dos problemas propostos. Pode-se afirmar que rAG é um procedimento adequado e barato computacionalmente, exige poucos recursos computacionais, para serem aplicados na resolução do problema FSP.

O rAG consegui bons resultados mesmo com procedimentos computacionais baratos, quando comparado com os algoritmos genéticos encontrados na literatura tais como AGChen e AGHC. O AGChen já começa com uma população inicial de boa qualidade, encontrada pelos métodos CDS e Danninbring. Testes realizados mostraram que uma taxa de *crossover* de 100% e sem mutação melhoraria o desempenho do algoritmo, com essas informações pode-se deduzir que o autor realizou vários testes até chegar a essa configuração do algoritmo. O AGHC utiliza o método HP que gera uma população inicial diversificada e a heurística 3-opt para realizar a mutação. O rAG não garante a geração de uma população inicial de boa qualidade, mas pôde-se constatar que mesmo assim ele ainda consegui bons resultados. Embora este fato possa ter um comportamento diferente para o uso desta técnica em outros problemas da classe dos POCP, neste experimento, ficou evidenciado que o alerta feito por Simões (1999) é de grande importância (os modelos computacionais que utilizam o paradigma dos AG se pareçam mais com os modelos biológicos).

O rAG pode ser aplicado a classe de problemas de otimização combinatória permutacional sem maiores dificuldades. Escolheu-se o FSP, como o representante dos POCP a ser estudado para uma aplicação da nossa técnica. Esta escolha não se deu ao acaso, visto que se trata de um dos problemas bastante estudado com instâncias já consolidadas na literatura. A modelagem de um problema de otimização combinatória permutacional, como um FSP, não prejudica em nada a generalidade com que ele poderia ser tratado aqui.

Trabalhos futuros podem ser realizados, tais como:

- Para diminuir o tempo de execução de rAG usaria-se o processamento paralelo ou distribuído, em vez de processamento seqüencial, tendo em vista que a busca leva muito tempo para encontrar uma nova solução melhor que a anteriormente encontrada. Isso não é um contra-senso em relação a proposta deste trabalho, já que uma meta-heurística não tem garantia de encontrar a solução ótima, então para isso pode ser utilizada alguma hibridização que melhore o desempenho. Poderia ser gasto um tempo a mais, uma vez que no procedimento é possível que a busca seja feita em paralela através dos diversos processadores, existentes e disponíveis para tal.
- Aplicar o rAG em outros problemas da classe dos POCP.
- Estudar o valor adequado para os parâmetros do procedimento, que não comprometa os recursos computacionais e gere boas soluções para o problema.

Agradecimentos
Os autores agradecem o apoio da Universidade Federal do Ceará, do CNPq (processo 302176/03-9) e da CAPES.

5. Bibliografia

Beasley, J.E. OR-Library: Distributing Test Problems by Eletronic Mail. *Journal of the Operations Research Society*, 41: 1069-1072, 1990.
Campbell, H.G., Dudek, R.A., Smith, M.L. A heuristic algorithm for the n-job, m-machine sequencing problem. *Management Science*, 16: B630–B637, 1970.
Chen, C.L., Vempati, V.S., ALJABER, N. An application of genetic for flow shop problems. *European Journal of Operational Research* 80, 389–396, 1995.
Conway, R.W., Maxwell, W.L., Miller, L.W. *Theory of scheduling. Reading*, MA: Addison-Wesley; 1967.
Garey, M.R., Johnson, D.S., Sethi, R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–29, 1976.
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.
Gupta, J.N.D. A Functional Heuristic Algorithm for the Flow-Shop Scheduling Problem. *Operational Research Quarterly* 22, 39-47, 1971.
Gupta, J.N.D., Stafford Jr, E.F. Flowshop scheduling research after five decades. *European Journal of Operational Research* 169, 699–711, 2006.
Haupt, R.L, Haupt, S.E. Pratical genetic algorithms. Wiley, USA, 1998.
Holland, J.H. *Adaptation in natural artificial systems*. University of Michigan Press, 1975.
Hundal, T.S. and Rajgopal, J. An Extension of Palmer's Heuristic for the Flow-Shop Scheduling Problem. *International Journal of Production Research* 26, 1119-1124, 1988.
Ignall, E. and Schrage, L.E. Application of Branch and Bound Technique to some Flow-Shop Problem. *Operations Research* 13, 400-412, 1965.
Johnson, S.M. Optimal two- and three-stage production schedules with setup times included. *Naval Research logistics Quarterly* 1, 61–68, 1954.
Moccellin, J. V. Um método Heurístico Híbrido Algoritmo Genético – Busca Tabu para a Programação Flow Shop Permutacional. Anais do XXI SBPO-Simpósio Brasileiro de Pesquisa Operacional, Juiz de Fora-MG, Brasil,
Nawaz, M., Enscore Jr., E.E. and Ham, I. A Heuristic Algorithm for the m-Machine n-Job Flow-Shop Sequencing Problem. *OMEGA* 11, 91-95, 1983.

Palmer, D.S. Sequencing jobs through a multistage process in the minimum total time - a quick method of obtaining a near optimum. *Operational Research Quartetly* 16: 101-107, 1965.

Potts, C.N. An Adaptive Branching Rule for the Permutation Flow-Shop Problem. *European Journal of Operational Research* 5, 19-25, 1980.

Reeves, C. R. *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, London, 1995.

Ruiz, R., Maroto, C., Alcaraz, J. Two newrobust genetic algorithms for the flowshop scheduling problem. *The International Journal the Management Science* (Omega), 34 : 461 – 476, 2006.

Selen, W.J. and Hott, D.D. A Mixed-Integer Goal Programming Formulation of theStandard Flow-Shop Scheduling Problem. *Journal of the Operational Research Society* 37, 1121- 1128, 1986.

Silva, J. L. C. e Soma, N. Y. Uma heurística para Problemas de Otimização Combinatória Permutacional. *Anais do XXXIII SBPO-Simpósio Brasileiro de Pesquisa Operacional*, Campos do Jordão-SP, Brasil, 2001.

Silva, J. L. C. e 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, 2006.

Simões, A. B. *Transposição: estudo de um novo operador genético inspirado biologicamente.* Dissertação de Mestrado em Engenharia de Informática – Departamento de Engenharia Informática – Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Coimbra, Portugal, 1999.

Taillard, E. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research* 64, 278-285, 1993.

Whitley, D., Starkweather, T. and Shaner, D. *The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination*. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, 350-372, 1991.

Wilson, J.M. Alternative Formulations of a Flow Shop Scheduling Problem. *Journal of the Operational Research Society* 40, 395-399, 1989.