Algoritmos genéticos para o problema de programação de tarefas em máquinas paralelas idênticas com dois critérios

José Elias Claudio Arroyo[1], Dalessandro Soares Vianna[2]

[1]Departamento de Informática – Universidade Federal de Viçosa - UFV
Viçosa – MG
[2]Departamento de Informática – Universidade Candido Mendes
Campos – RJ
jarroyo@dpi.ufv.br, dalessandro@ucam-campos.br

Resumo: *Este artigo aborda o problema de escalonamento de tarefas em máquinas paralelas idênticas com tempos de preparação dependentes da seqüência. Este problema consiste em determinar a ordem de processamento de um conjunto de tarefas em um conjunto de máquinas, minimizando simultaneamente dois critérios: tempo de finalização do processamento das tarefas (makespan) e atraso total das tarefas com relação a suas datas de entrega. Este problema possui um conjunto de soluções denominadas Pareto-ótimas ou eficientes. O problema é resolvido aproximadamente usando três algoritmos genéticos diferentes que usam estratégias de busca local. Resultados computacionais mostram que os algoritmos genéticos apresentam um bom desempenho em termos de qualidade de soluções e tempo de execução.*

Abstract: *This paper addresses the identical parallel machine scheduling problem with sequence dependent setup times. This problem consists in determining the job order sequence on each machine minimizing simultaneously two criteria: maximum completion time (makespan) and the total job tardiness with relation to due dates. This problem has a set of solutions denominated Pareto optimal or efficient. The problem is approximately solved using three different genetic algorithms with local search strategies. Computational results show that the genetic algorithms present good performance both in terms of the quality of solutions and execution time.*

1. Introdução

Os problemas de programação de tarefas encontram-se intimamente ligados ao planejamento e programação da produção nas indústrias. Estes problemas, além de matematicamente desafiadores, são extremamente difíceis de resolver. A explosão combinatória decorrente da necessidade de se examinar as várias alternativas de escalonamento existentes nesses problemas torna difícil e custosa a obtenção de uma solução ótima ou aproximada.

Em otimização multicritério, ao contrário da otimização de um único critério (mono-critério), a solução esperada é composta por um conjunto de soluções de equilíbrio, denominados eficientes ou Pareto-ótimas, caracterizadas por não serem "dominadas" por nenhuma outra solução do conjunto factível. A partir deste conjunto

pode-se escolher a solução de melhor compromisso de acordo com as preferências fixadas pelo decisor (*decision maker*).

Em problemas combinatórios multicritério, geralmente, a otimização individual de cada critério é NP-dificil. O grau de dificuldade para tratar estes problemas aumenta ainda mais, quando se trata em determinar todas as soluções Pareto-ótimas, devido ao conflito existente entre os critérios. Os métodos mais adequados para viabilizar a resolução destes problemas são os métodos heurísticos/metaheurísticos. Estes métodos, quando bem desenvolvidos e adaptados aos problemas de otimização multicritério, são capazes de apresentar soluções aproximadas de boa qualidade em tempo polinomial [9].

Muitos pesquisadores propuseram metaheurísticas para resolver problemas de otimização multicritério. Por exemplo, algoritmos genéticos ([3], [15]), métodos de busca local baseados em busca tabu ([2], [10], [12]) e *simulated annealing* ([6]). A maioria esmagadora das publicações de metaheurísticas para problemas de otimização multicritério são baseadas em algoritmos genéticos [15]. Esta preferência se deve ao fato que os algoritmos genéticos trabalham com uma população de soluções e que podem conter informação sobre várias regiões do espaço de busca e, portanto, estes oferecem maiores possibilidades para encontrar uma boa aproximação do conjunto de soluções Pareto-ótimas.

Para o problema programação de tarefas em máquinas paralelas (PTMP), recentemente foram propostas algumas metaheurísticas para a otimização de um único critério: *simulated annealing* [16], algoritmos genéticos ([17], [19]), busca tabu ([1], [4]). Os trabalhos da literatura enfatizam a importância em considerar aspectos como, *makespan*, datas de entrega e tempos de preparação dependentes da seqüência (*setup-times*).

Neste trabalho procura-se encontrar aproximações de soluções Pareto-ótimas do problema de escalonamento de tarefas em máquinas paralelas idênticas, minimizando dois critérios: *makespan* (C_{max}) e atraso total das tarefas em relação às suas datas de entrega (T). C_{max} está relacionado com a utilização máxima das máquinas, enquanto T está associada com liberação das tarefas nas respectivas datas de entrega (on-time-delivery). Recentemente uma revisão bibliográfica apresentada em [22], sobre abordagens multicritério de problemas de escalonamento de tarefas, mostra que a pesquisa é principalmente focada em problemas de escalonamento em máquinas simples ([8]). Para o problema multicritério de PTMP, poucos trabalhos foram publicados. Em [21] é apresentada uma metaheurística busca tabu para o problema PTMP minimizando o *makespan* e o atraso máximo das tarefas. Em [5] é desenvolvido um algoritmo genético para o problema PTMP para a minimização de três critérios: *makespan*, soma ponderada dos tempos de finalização das tarefas e soma ponderada dos atrasos das tarefas.

Neste artigo analisa-se a eficiência de três algoritmos genéticos com busca local (algoritmos híbridos) para resolver o problema PTMP. O primeiro algoritmo genético multicritério (AGMC1), aplicado previamente para o problema de *flowshop scheduling* [3], usa o conceito de dominância de soluções. Durante o processo da busca do algoritmo, uma solução será considerada "apta" se, ela não é dominada por outras soluções e ela está localizada em uma região pouco explorada. Neste algoritmo também é usada uma estratégia de elitismo que consiste em adicionar na população algumas soluções não dominadas encontradas até o momento. O objetivo é que estas soluções não dominadas participem do processo de cruzamento tentando gerar novas soluções com características similares.

O segundo algoritmo genético multicritério (denotado por AGMC2), é proposto por Jaszkiewicz (2002) [14]. Jaszkiewicz aplicou o algoritmo genético para resolver o problema do Caixeiro Viajante com múltiplos critérios e os resultados obtidos foram superiores quando comparados com os resultados de outro algoritmo genético híbrido proposto por Ishibuchi e Murata (1998) [13]. Os algoritmos genéticos de Jaszkiewicz e Ishibuchi-Murata possuem uma idéia em comum, que consiste em usar diferentes vetores de pesos (preferências pelos critérios) e gerar diferentes soluções otimizando uma função utilidade definida pela combinação ponderada dos critérios.

É proposto um terceiro algoritmo genético (denotado por AGMC3) que é uma versão melhorada do algoritmo AGMC2. Neste algoritmo é proposto um procedimento de intensificação que consiste em recombinar as soluções melhoradas pela busca local com as soluções não dominadas encontradas até o momento (soluções de elite).

O artigo é organizado da seguinte maneira. A Seção 2 contém a descrição do problema abordado e conceitos usados em otimização multicritério. Na Seção 3 é apresentados os três algoritmos implementados. Na Seção 4 estão os operadores genéticos usados nos algoritmos. Os testes computacionais realizados estão na Seção 5, e na ultima seção estão as conclusões do trabalho.

2. O Problema de programação de tarefas em máquinas paralelas com múltiplos critérios

Sejam, um conjunto de n tarefas e um conjunto de m máquinas idênticas ($m < n$). Cada tarefa i deve ser processada em uma máquina em um tempo fixo p_i. Cada tarefa i possui uma data de entrega $d_i \geq 0$. Cada máquina pode processar somente uma tarefa de cada vez e não é permitido interromper o processamento de qualquer das tarefas, uma vez que tenham sido iniciados. É considerado tempo de preparação de máquina (*setup time*) S_{ij}, definido como o tempo gasto para fazer a manutenção de uma máquina após de finalizar o processamento de uma tarefa i e antes de iniciar com o processamento de outra tarefa j ($S_{ij} \neq S_{ji}$).

O atraso de uma tarefa i é definido como $T_i = \max\{C_i - d_i, 0\}$, onde C_i é o tempo de finalização da tarefa i. Os critérios considerados são:
i) Minimizar o tempo de finalização do processamento das tarefas, $C_{max} = \max \{C_i\}_{1 \leq i \leq n}$,
ii) Minimizar o atraso total das tarefas, $T = \sum_{i=1}^{n} T_i$.

Uma solução viável x deste problema é formada por m seqüências de tarefas $[M_1,...,M_m]$ (onde M_k é a seqüência de tarefas a serem processadas na máquina k) e por um vetor cujas componentes são os valores dos critérios: $(f_1(x), f_2(x)) = (C_{max}, T)$.

Dado um vetor de r critérios ou funções objetivos $f = (f_1, \ldots, f_r)$ definidas em um conjunto discretos Ω. Um problema de otimização multicritério pode ser formulado da seguinte maneira:

Minimizar $f(x) = (f_1(x) = z_1, \ldots, f_r(x) = z_r)$
Sujeito a $x \in \Omega$

Definição. Um ponto z *domina* z' se $z_j = f_j(x) \leq z'_j = f_j(x')$, $\forall j$ e $z_j < z'_j$ para pelo menos um j. Uma solução $x^* \in \Omega$ é Pareto-ótima (ou *eficiente*) se ela não é dominada por nenhuma outra solução, isto é, não existe $x \in \Omega$ tal que $z = f(x)$ domine $z^* = f(x^*)$.

Por exemplo, sejam três soluções (escalonamentos) s_1, s_2 e s_3 com os valores de critérios (70, 77), (62, 90) e (69, 63) respectivamente. Observa-se que s_3 domina s_1; s_2 e s_3 não são dominados (s_2 possui menor valor de C_{max} e s_3 possui menor valor de T). Neste caso, as soluções s_2 e s_3 seriam as soluções Pareto-ótimas ou eficientes.

3. Algoritmos genéticos para resolver o problema abordado

Nesta seção são apresentados os algoritmos genéticos propostos por Arroyo e Armentano, (2005) e Jaszkiewicz, (2002), respectivamente denotados por AGMC1 e AGMC2. Também é apresentada uma versão melhorada do algoritmo AGMC2, que é denotado por AGMC3.

3.1. Algoritmo genético multicritério 1 (AGMC1)

Neste algoritmo, para atribuir o valor de *fitness* às soluções da população P_t, antes, é necessário classificá-las segundo o nível de dominância das soluções. É usada a técnica de ordenação por dominância (*Nondominated Sorting Technique*) (veja [7]). Esta técnica classifica a população em K conjuntos F_1, F_2,..., F_K, onde as soluções de cada conjunto F_i, $i = 1, 2, ...,K$, possuem o mesmo nível de dominância.

Após de classificar a população, um valor de *fitness* deve ser atribuído às soluções de cada conjunto F_k. O *fitness* é calculado de maneira que as "melhores" soluções possuam maiores valores de *fitness*. É claro que os valores de *fitness* das soluções no conjunto F_k devem ser maiores que os valores de *fitness* das soluções no conjunto F_j, $j > k$. Os valores de *fitness* atribuídos às soluções do conjunto F_k são calculados através de uma estratégia que tenta preservar a dispersão da população. Para cada solução $z \in F_j$, determina-se a estimativa do número de pontos localizados em redor dele. Esta estimativa é denominada de distância de *crowding* (veja [3]).

O valor de *fitness* de uma solução $x \in F_j$ é a soma da sua distância de *crowding* e do máximo *fitness* em F_{j+1}, portanto as soluções em F_j são mais aptas que as soluções em F_{j+1}. Note que para problemas de minimização e maximização o *fitness* é maximizado, ou seja, valores altos de *fitness* correspondem a grandes probabilidades de seleção. Após de calcular os *fitness* de todas as soluções da população P_t, são executados os operadores de seleção, recombinação e mutação construindo uma nova população de N soluções filhos.

A estratégia de elitismo usada no algoritmo AGMC1 consiste em selecionar N_{elite} soluções do conjunto de soluções não dominadas ND para que sejam adicionadas na população P_t. Estas N_{elite} soluções de elite participam do processo de classificação da população e, conseqüentemente, da geração de novas soluções através dos operadores genéticos.

O algoritmo AGMC1 usa também uma técnica de busca local com o objetivo de melhorar as soluções da população. O método é baseado no conceito de dominância de Pareto e explora em paralelo várias regiões do espaço de soluções. A busca começa com o conjunto de soluções não dominadas S da população e a partir de cada solução x deste conjunto gera-se uma vizinhança $N(x)$. Constrói-se então o próximo conjunto S' das soluções vizinhas não dominadas por S onde S' é um subconjunto de $\cup_{x \in S} N(x)$. Caso $S' \neq \varnothing$, o processo se repete a partir deste novo conjunto.

Se o conjunto S (ou S') possui mais de N_d soluções, este conjunto deve ser reduzido a N_d soluções e estas soluções devem estar espalhadas no respectivo conjunto ou fronteira. Por tanto, sempre serão exploradas, no máximo, N_d soluções em paralelo. Para evitar que a busca local consuma a maior parte do tempo computacional do

algoritmo genético, sugere-se acionar a busca local a cada β iterações do algoritmo genético. O pseudocódigo do AGM1 é apresentado em [3].

3.2. Algoritmo genético multicritério 2 (AGMC2)

No algoritmo AGMC2, proposto por Jaszkiewicz (2002) [14], o *fitness* é determinada como sendo a soma ponderada dos critérios (função utilidade linear):

$f(x, w) = \sum_{i=1}^{r} w_i \cdot f_i(x)$, onde $w = (w_1,.., w_r)$ é o vetor de pesos atribuídos a os r critérios.

Esta função utilidade satisfaz um importante papel na caracterização de soluções Pareto-ótimas. Cada função utilidade linear possui um ótimo no conjunto de soluções Pareto-ótimas. E, encontrar todas as soluções Pareto-ótimas é equivalente a encontrar os ótimos de todas as funções utilidade $f(x,w)$ [20].

Na implementação do algoritmo genético AGMC2, três conjuntos de soluções são definidos:
ND: conjunto das soluções não dominadas ou conjunto aproximado das soluções Pareto-ótimas (saída do algoritmo).
P: população dinâmica de soluções
*P**: sub-população temporária formada pelas *N* melhores soluções de *P*.

O algoritmo genético começa gerando *Psize* soluções iniciais. Todas as soluções da população *P* são melhoradas através de um procedimento de busca local. A cada iteração do algoritmo, gera-se um vetor de pesos $w = (w_1, w_2)$ para cada solução de *P* e calcula-se o valor da função utilidade $f(x,w)$. Em seguida cria-se uma sub-população temporária *P** formadas pela *N* (*N*<*Psize*) melhores soluções de *P*. Duas soluções x_1 e x_2 de *P** são selecionadas (soluções pais). Estas soluções são recombinadas gerando uma solução filho x_3. Logo, x_3 é melhorada através de uma busca local obtendo x_3'. Se x_3' é melhor que a pior solução de *P**, então ela será inserida em *P*. Vale ressaltar que, a busca local deste algoritmo genético é baseada na otimização da função utilidade linear, e a cada iteração a seleção das soluções pais é feita sempre utilizando pesos diferentes (w_1, w_2).

Para a população *P*, define-se um tamanho máximo N_{max} (>*Psize*). Esta população é implementada na forma de uma fila circular. Novas soluções encontradas e melhoradas são inseridas no final da fila. Caso a fila fique cheia (com N_{max} soluções), a solução mais antiga (localizada no início da fila) é removida. A seguir apresenta-se um pseudocódigo do algoritmo AGMC2 para o caso de dois critérios.

***Algoritmo* AGMC2**
Entrada: *N, Psize,* N_{max}
Saída: *ND* (*conjunto de soluções não dominadas*)
0) Faça $ND = \varnothing$ e $P = \varnothing$.
- ***Geração da população inicial*:**
1) Usando uma heurística construtiva gere duas soluções: x (que minimiza C_{max}) e y (que minimiza T).
2) Faça: $P = P \cup \{x, y\}$ e *ND* = soluções não dominadas de $ND \cup \{x, y\}$.
3) Construa aleatoriamente (*Psize* –2) soluções factíveis a serem inseridos em *P*.
4) Para cada solução x_i de *P* faça os seguintes passos:
   4.1) Gere aleatoriamente um vetor de pesos $w = (w_1, w_2)$.
   4.2) Calcule a função utilidade $f(x_i, w)$.
   4.3) Melhore o valor de $f(x_i, w)$ aplicando busca local à solução x_i e obtendo x_i^*.
   4.4) Adicione x_i^* a *P*: $P = P \cup \{ x_i^* \}$.
   4.5) Atualize *ND* com x_i^* : *ND* = soluções não dominadas de $ND \cup \{ x_i^* \}$.

- ***Loop principal*:**

5) Enquanto a condição de parada não for satisfeita, execute os próximos passos:
6) Gere aleatoriamente um vetor de pesos $w = (w_1, w_2)$.
7) Calcule $f(x, w)$, $\forall x \in P$.
8) Selecione as N melhores soluções diferentes de P, formando a população P^*.
9) Selecione aleatoriamente duas soluções x_1 e x_2 de P^*.
10) Recombine x_1 e x_2 formando x_3.
11) Melhore a solução x_3 aplicando uma busca local e obtendo x'_3.
12) ND = soluções não dominadas de $ND \cup \{x'_3\}$.
13) Se x'_3 é melhor que a pior solução de P^* e $x'_3 \neq x$, $\forall x \in P^*$, então
    13.1) Se $|P| < N_{max}$ então adicione x'_3 a P.
    13.2) Senão, remova a solução mais antiga de P e adicione x'_3 a P.

3.3. Algoritmo genético multicritério 3 (AGMC3)

Neste trabalho é proposta uma versão melhorada do algoritmo AGMC2 o qual é denotado por AGMC3. O AGMC3 é implementado adicionando em AGMC2 um procedimento de intensificação que consiste em recombinar a solução x'_3 obtida no Passo 11 com cada uma das soluções não dominadas $x \in ND$ (encontradas até o momento) obtendo diferentes soluções x_4. As soluções x_4 também são melhoradas através da busca local e estas soluções também podem ser inseridas na população P. Vale ressaltar que a busca local é baseada na otimização da função utilidade linear, e a cada iteração a seleção das soluções pais é feita sempre utilizando pesos diferentes (w_1, w_2). O algoritmo AGMC3 possui os Passos 0 a 13 igual ao AGMC2, sendo que AGMC3 possui três Passos a mais que correspondem a uma fase de intensificação:

- ***Fase de Intensificação:***

14) Defina ND' = $\{x'_3\}$ um conjunto auxiliar de soluções não dominadas.
15) Para cada solução $x \in ND$ faça:
    15.1) Recombine x e x'_3 obtendo x'.
    15.2) Calcule $f(x', w)$ e aplique a busca local a x' obtendo x''.
    15.3) ND' = soluções não dominadas de $ND' \cup \{x''\}$.
    15.4) Se $f(x'', w) < f(x'_3, w)$ então
        Insira x'' em P de maneira similar aos passos 13.1 e 13.2.
        Faça $x'_3 = x''$.
16) ND = soluções não dominadas de $ND \cup ND'$.

4. Operadores genéticos dos algoritmos genéticos

Para representar um escalonamento de n tarefas em m máquinas paralelas é utilizado um vetor a de tamanho $(n+m-1)$, onde são armazenados números de 1 a n (correspondentes às n tarefas) e $(m-1)$ números -1 que são pontos de corte no cromossomo que define as seqüências de tarefas atribuídas a cada máquina. Por exemplo, para $n = 10$ e $m = 3$, o cromossomo [2, 8, 10, -1, 4, 1, 6, 5, -1, 7, 3, 9] corresponde às seqüências de tarefas M_1 = {2, 8 10}, M_2 = {4, 1, 6, 5} e M_3 = {7, 3, 9}, processadas nas máquinas 1, 2 e 3, respectivamente. Devido aos tempos de *setup*, em cada máquina é considerada a ordem na quais as tarefas são processadas.

A população inicial dos algoritmos genéticos é formada por *Psize* soluções, das quais duas soluções são construídas heuristicamente e (*Psize* –2) soluções são geradas aleatoriamente. Foram implementadas duas heurísticas diferentes para a minimização do *makespan* e do atraso das tarefas. A heurística para a minimização do *makespan* começa inserindo as m tarefas com os menores tempos de processamento em cada uma das m máquinas. Em seguida, são inseridas as demais tarefas selecionando primeiro a tarefa que produza o menor instante de término possível em uma determinada máquina. Para a

minimização do atraso das tarefas, foi implementada a heurística proposta em [11], chamada "*Sequential Assignment Algorithm*". Esta heurística é composta por duas fases. Na primeira fase, constrói-se uma lista de tarefas ordenadas por valores não decrescentes do parâmetro ($d_i - p_i$) (diferença entre a data de entrega da tarefa e seu tempo de processamento). Na segunda fase, cada uma das tarefas, tomadas de acordo com a ordenação realizada, é inserida em uma das máquinas de forma que seu instante de término seja o menor possível.

Nos três algoritmos genéticos, para recombinar duas soluções pais x_1 e x_2, foi implementado o operador *order crossover* (OX) [18]. Inicialmente um fragmento aleatório de x_1 é copiado no filho x_3. As posições vazias em x_3 são preenchidas seqüencialmente com elementos de x_2 ainda não inseridos em x_3. Vale lembrar que cada solução deve conter (m–1) números -1. Um exemplo deste operador é mostrado na Figura 1.

x_1 :	2	8	10	**-1**	4	1	6	5	**-1**	7	3	9
x_2 :	5	7	4	8	**-1**	10	9	2	**-1**	1	6	3
x_3 :	5	7	10	**-1**	4	1	6	8	**-1**	9	2	3

Figura 1. Operador *order crossover*.

No AGMC1, para selecionar os pais x_1 e x_2 foi usado o método da roleta (*roulette wheel*) [18]. A mutação implementada no algoritmo AGMC1 consiste na troca de tarefas. As tarefas são selecionadas aleatoriamente numa mesma máquina ou em máquinas diferentes. Nos algoritmos AGMC2 e AGMC3 não são usadas mutação. Nestes algoritmos o uso de diferentes vetores de pesos gerados aleatoriamente introduzem um mecanismo de diversificação [14].

Na fase da busca local dos algoritmos genéticos, a vizinhança de uma solução x é gerada através da inserção de tarefas. Uma solução vizinha é gerada inserindo uma tarefa i (localizada na posição k da seqüência) em outra posição da seqüência. Nos algoritmos genéticos AGMC1 e AGMC2 são analisados todos os possíveis vizinhos de uma solução x em sua vizinhança $N(x)$. Esta estratégia gera uma vizinhança de tamanho ($n^2 + m$). Nos algoritmos AGMC2 e AGMC3 o vizinho candidato somente é aceito se ele melhorar estritamente o valor da melhor solução até então obtida pela busca local. Dessa forma, a busca pára quando um mínimo local seja encontrado. É claro que, a avaliação de todas as soluções geradas pela vizinhança completa de inserção é custosa. Com o propósito de evitar que a busca local consuma muito tempo computacional, nos três algoritmos genéticos é limitado o número máximo de iterações da busca local (*Niter*).

5. Testes computacionais

Nesta seção são comparados os desempenhos dos algoritmos genéticos AGMC1, AGMC2 e AGMC3. Estes Os três algoritmos foram implementados usando a linguagem de programação C++ e executados em um computador com processador Pentium IV de 2.26 GHz e 512 de RAM.

5.1. Geração de instâncias

A geração de instâncias (problemas testes) para o problema abordado segue o mesmo esquema apresentado em [1]. Os parâmetros de cada tarefa são todos números inteiros gerados a partir de uma distribuição uniforme nos seguintes intervalos: tempos de

processamento [1,100]; tempos de *setup* [0,20]; datas de entrega [0,αP], onde $P = \frac{1}{m}\sum_{i=1}^{n} p_i$ e α é um parâmetro que determina a restritividade das datas de entrega. Os valores utilizados para α foram 0,6; 0,8; 1,0 ou 1,2. Foram considerados problemas com n = 20, 40, 50, 80 e 100 e m = 2, 4, 6 e 8. Para cada par de valores (m, n) foram gerados 20 problemas, 5 para cada valor de α. No total foram gerados 400 problemas.

5.2. Parâmetros dos algoritmos genéticos

Para fixar os valores dos parâmetros usados nos algoritmos genéticos foram gerados diversos testes computacionais. Os melhores resultados foram obtidas usando os seguintes parâmetros:

AGMC1	AGMC2 e AGMC3
• Tamanho da população: *Psize* = 100; • Nº máximo de soluções de elite: N_{elite} = 10; • Probabilidade de recombinação: p_R = 0,8; • Probabilidade de mutação: p_M = 0,1; • Ativação da busca local: A cada β = 4 iterações; • Nº máximo de soluções a serem exploradas pela busca local (busca em paralelo): N_d = 6; • Nº máximo de iterações da busca local: *Niter* = 12.	• Tamanho da população inicial: *Psize* = 80; • Tamanho máximo da população dinâmica (fila): N_{max} = 250; • Tamanho da sub-população temporária: N = 20; • Probabilidade de recombinação: p_R = 1,0; • Nº máximo de iterações da busca local: *Niter* = 12.

Com a finalidade de comparar os três algoritmos, é utilizado um critério de parada que consiste na avaliação de um número máximo de soluções. Para o problema abordado, o número máximo de soluções avaliadas por cada algoritmo foi $3500 \times n^2$, onde n = número de tarefas.

5.3. Comparação dos resultados

Para avaliar e comparar as soluções geradas pelos algoritmos genéticos são usadas as mesmas medidas utilizadas em [3]: i) número de soluções não dominadas obtidas por cada algoritmo, ii) distância média (D_{med}) das soluções obtidas com relação ao conjunto de referência formadas pelas soluções não dominadas obtidas pelos três algoritmos.

Para todos os conjuntos de instâncias, analisa-se o desempenho dos três algoritmos genéticos. Na Tabela 1, para cada tamanho de problema, mostra-se o número de soluções de referência (ou total de soluções não dominadas obtidas pelos três algoritmos genéticos) - $|R|$ e, para cada um dos algoritmos mostra-se o número total de soluções obtidas (NTS), o número de soluções não dominadas (NSND) e a qualidade das soluções dada pela distancia média D_{med}. Para as 400 instâncias testadas, em total foram encontradas 4355 soluções não dominadas das quais os algoritmos AGMC1, AGMC2 e AGMC3 obtiveram, respectivamente, 2450, 1392 e 2276 soluções. A Tabela 1 mostra que AGMC1 obteve um número maior de soluções não dominadas em todos os tamanhos de problemas (exceto em $n \times m = 20 \times 6$, onde AGMC3 gerou 5 soluções a mais do que AGMC1). Para todos os tamanhos de problema, em termos de qualidade de soluções geradas, observa-se que AGMC1 é superior a AGMC2 e AGMC3. Note que a versão melhorada de AGMC2, AGMC3, é muito superior a AGMC2.

Devido ao mesmo critério de parada usado nos três algoritmos, os tempos computacionais gastos pelos algoritmos foram bem próximos. Por exemplo, para problemas com 100 tarefas e 8 máquinas a média dos tempos gastos pelos algoritmos AGMC1, AGMC2 e AGMC3 foram 83s, 75s e 77s, respectivamente.

Tabela 1. N^o de soluções e qualidade das soluções obtidos pelos três algoritmos.

$n \times m$	$\|R\|$	AGMC1			AGMC2			AGMC3		
		NTS	NSND	D_{med}	NTS	NSND	D_{med}	NTS	NSND	D_{med}
20×2	315	226	152	0,012	199	119	0,049	218	141	0,030
20×4	168	122	81	0,013	152	69	0,068	119	70	0,037
20×6	171	125	84	0,016	125	58	0,052	129	89	0,018
20×8	156	116	85	0,021	116	55	0,050	121	72	0,048
40×2	275	227	156	0,013	212	76	0,057	231	146	0,033
40×4	202	155	116	0,012	177	60	0,087	151	111	0,042
40×6	194	145	87	0,013	150	74	0,068	148	77	0,033
40×8	180	158	82	0,024	156	54	0,058	147	74	0,044
50×2	289	272	125	0,008	250	111	0,067	260	114	0,038
50×4	198	176	136	0,003	182	51	0,080	183	119	0,023
50×6	195	160	99	0,017	177	82	0,073	167	88	0,037
50×8	174	152	105	0,005	156	54	0,071	160	92	0,025
80×2	269	234	184	0,009	215	62	0,160	223	169	0,032
80×4	206	209	129	0,003	182	49	0,080	199	126	0,033
80×6	201	168	93	0,017	177	81	0,073	170	86	0,047
80×8	188	172	102	0,005	156	53	0,071	160	95	0,025
100×2	279	258	197	0,018	222	55	0,183	251	190	0,042
100×4	244	198	170	0,016	194	60	0,172	204	163	0,046
100×6	240	206	155	0,004	178	78	0,098	194	152	0,024
100×8	211	198	112	0,024	169	91	0,088	191	102	0,063
	Total: **4355**	Total: 3677	Total: **2450**	Média: 0,012	Total: 3545	Total: **1392**	Média: 0,085	Total: 3626	Total: **2276**	Média: 0,036

6. Conclusões

Neste artigo foi abordado o problema de programação de tarefas em máquinas paralelas (PTMP) otimizando dois critérios. Foram comparados os desempenhos de três algoritmos genéticos híbridos, com estratégias diferentes, na geração de soluções não dominadas ou soluções eficientes aproximadas. Os algoritmos AGMC1 e AGMC2 propostos na literatura ([3] e [14]) foram adaptados para o problema PTMP. Neste trabalho é proposta uma versão melhorada do algoritmo AGMC2, aqui denominado AGMC3, que usa uma estratégia de intensificação baseada na recombinação das soluções obtidas pela busca local com as soluções não dominadas encontradas até o momento. Através de testes computacionais sobre uma variedade significativa de problemas, os resultados mostram que os algoritmos AGMC1 e AGMC3 possuem melhores desempenhos, em termos de número de soluções obtidas e qualidade.

Referências

[1] Armentano, V. A., Yamashita, D. S. (2000) Tabu Search for Schedulling on Identical Parallel Machines to Minimizing Mean Tardiness. Journal Of Intelligent Manufacturing. , Vol.11, No.5, p.453- 460.

[2] Armentano V.A. and Arroyo J.E.C., (2004) An Application of a Multi-Objective Tabu Search Algorithm to a Bicriteria Flowshop Problem. *Journal of Heuristics*. Vol. 10, No. 5, pp. 463-481.

[3] Arroyo J.E.C. and Armentano V.A., (2005) A Genetic Local Search Algorithm for Multi-Objective Flowshop Scheduling Problems. *European Journal of Operational Research*, Vol. 167, No.3, pp. 717-738.

[4] Bilge, U., Kirac, F., Kurtulan, M. and Pekgun, P., (2004) A tabu search algorithm for parallel machine total tardiness problem. *Computers and Operations Research*, Vol.31, No.3, pp. 397-414.

[5] Cochran J. K., Horng S-M and Fowler J.W. (2003) A Multi-Population Genetic Algorithm to Solve Multi-Objective Scheduling problems for Parallel Machines. *Comp & Operations Research*, Vol.30 No.7 pp 1087-1102.

[6] Czyzak P. and Jaszkiewicz A. (1998) Pareto Simulated Annealing – a metaheuristic technique for multiple objective combinatorial optimization, *Journal of Multi-Criteria Decision Analysis*, Vol. 7, pp. 34-47.

[7] Deb, K.; Agrawal, S.; Pratab, A. & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India.

[8] Dileepan P. and Sen T. (1988) Bicriterion static scheduling research for a single machine, *OMEGA*, Vol. 16, pp. 53-59.

[9] Ehrgott M. (2000) Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, Vol. 7, pp. 5-31.

[10] Gandibleux, X. and Fréville, A. (2000). Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: The two objectives case. *Journal of Heuristics*, Vol. 6, pp.361-383.

[11] Guinet, A., (1995) Scheduling Independent Jobs on Uniform Parallel Machines to Minimize Tardiness Criteria, *Journal of Intelligent Manufacturing*, Vol. 6, pp. 95-103.

[12] Hansen, M.P. and Jaszkiewicz, A. (1998). Evaluating the quality of approximations to the nondominated set. Technical Report, Institute of Mathematical Modeling, Technical University of Denmark.

[13] Ishibuchi, H. and Murata, T. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man and Cybernetics-part C*, Vol. 28, pp. 392-403.

[14] Jaskiewicz, A. (2002). On the performance of multiple objective genetic local search on the 0/1 knapsack problem: A comparative experiment. *IEE Transaction on Evolutionary Computation,* Vol. 6, No 4, pp. 402-412.

[15] Jones D.F., Mirrazavi S.K. and Tamiz M. (2002) Multi-objective meta-heuristics: An overview of the current state-of-art. *European Journal of Operational Research,* Vol.137, pp. 1-19.

[16] Lee, Y.H., Bhaskaram, K. and Pinedo, M. (1997) A heuristic to minimize the total weighted tardiness with sequence-dependent setup times. *IIE Transactions*, Vol.29, No. 1, pp. 45-52.

[17] Mendes, A., Müller, F., França P. and Moscato P. (2002) Comparing Meta-Heuristic Approaches for Parallel Machine Scheduling Problems, *Production Planning & Control*, Vol. 13, No. 2, pp. 143-154.

[18] Michalewicz Z. (1996) Genetic Algorithms + Data Structures = Evolution Programs. *Thir Edition*, *Springer-Verlag Berlin Heidelberg New York.*

[19] Min, L. and Cheng, W. (1999) A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines, *Artificial Intelligence in Engineering*, Vol. 13, No. 4, pp.399-403.

[20] Steuer, R.E. (1986), Multiple Criteria Optimization – Theory, Computation and Application. *Wiley*, New York.

[21] Suresh V. and Chaudhuri D. (1996) Bicrteria Scheduling Problem for Unrelated Parallel Machines. *Computers Ind. Engng*, Vol. 130, No.1, pp. 77-82.

[22] T'Kindt V. and Billaut J.C. (2001) Multicriteria Scheduling Problems: A Survey. *RAIRO Operational Research* , Vol. 35, pp. 143-163.