Learning and Nonlinear Models - Revista da Sociedade Brasileira de Redes Neurais (SBRN), Vol. 4, No. 1, pp. -32-42, 2006

METAHEURÍSTICA HÍBRIDA ALGORITMO GENÉTICO-*CLUSTERING SEARCH* PARA A OTIMIZAÇÃO EM SISTEMAS DE PRODUÇÃO *FLOW SHOP* PERMUTACIONAL

Geraldo Ribeiro Filho
Faculdade Bandeirantes de Educação Superior
R. Jose Correia Gonçalves, 57 – CEP: 08675-130, Suzano - SP
geraldorf@uol.com.br

Marcelo Seido Nagano
Escola de Engenharia de São Carlos – USP
Av. Trabalhador São-Carlense, 400 – CEP: 13566-590, São Carlos – SP
drnagano@usp.br

Luiz Antonio Nogueira Lorena
Laboratório Associado de Computação e Matemática Aplicada - INPE
Av. dos Astronautas, 1758 – CEP: 12227-010, São José dos Campos – SP
lorena@lac.inpe.br

RESUMO

Este artigo trata do problema de programação de operações em um ambiente de produção *Flow Shop*, tendo como objetivo minimizar o estoque em processamento. Uma nova metaheurística híbrida, Algoritmo Genético – *Cluster Search*, é proposta para solução do problema. Por meio de experimentação computacional, o desempenho do método proposto é avaliado e comparado com os melhores resultados reportados na literatura. Os resultados experimentais mostraram a superioridade do novo método para o conjunto de problemas tratados em relação à qualidade das soluções obtidas.

Palavras chave: *Flow shop* permutacional, Estoque de processo, Metaheurística, Algoritmo genético, *Clustering search*.

ABSTRACT

This paper deals with the Permutation Flow Shop scheduling problem with the objective of minimizing total flow time, therefore reducing in-process inventory. A new hybrid metaheuristic, Genetic Algorithm - Cluster Search, is proposed for the scheduling problem solution. The proposed method is compared with the bests results reported in the literature. Experimental results show that the new method provides better solutions regarding the solution quality.

***Keywords*: *Permutation Flow shop, In-process inventory, Metaheuristic, Genetic algorithm, Clustering search*.**

1. Introdução

O problema tradicional de Programação *Flow Shop* é um problema de produção onde um conjunto de *n* tarefas deve ser processado, na mesma seqüência, por um conjunto de *m* máquinas. Quando a ordem de processamento em todas as máquinas for a mesma, tem-se o ambiente de produção *Flow Shop* Permutacional, onde o número de possíveis programações para *n* tarefas é *n*!.

O problema consiste em obter uma seqüência das tarefas que otimiza uma determinada medida de desempenho. Nos modelos para solução do problema, as medidas usuais referem-se à minimização da duração total da programação (*makespan*), associada à utilização eficiente dos recursos produtivos, e à minimização da soma dos tempos de fluxo das tarefas (*total flow time*) associados à redução do estoque em processamento, sendo esta última a medida adotada neste trabalho.

Este problema de programação da produção é *NP-hard* (GAREY *et al*., 1976 e RINNOOY KAN, 1976), e portanto, a busca por uma solução ótima apresenta uma importância mais teórica do que prática, direcionando as pesquisas para o desenvolvimento de métodos heurísticos e metaheuristicos, sendo os mais recentes descritos sucintamente a seguir.

2. Métodos Heurísticos para Minimização da Soma dos Tempos de Fluxo das Tarefas

Atualmente na literatura os métodos heurísticos podem ser divididos em duas grandes classes: os métodos heurísticos construtivos e os melhorativos.

Na literatura os primeiros métodos heurísticos construtivos para o problema foram as heurísticas propostas por Gupta (1972), Miyazaki *et al*. (1978), Rajendran e Chaudhuri (1991) e Rajendran (1993).

Rajendran e Chaudhuri (1991) desenvolveram três heurísticas que obtiveram melhor desempenho que os métodos heurísticos propostos por Gupta (1972) e Miyazaki *et al*. (1978), no que se refere à qualidade de solução e ao esforço computacional.

Ho (1995) propôs uma heurística composta de diferentes iterações no processo de melhoria de uma solução inicial, a partir da obtenção de um ótimo local por permutação de pares de tarefas adjacentes, melhorando posteriormente a solução por movimentos de inserção de tarefas. Tal método heurístico apresentou desempenho melhor que os anteriores reportados na literatura, embora, como esperado, com desvantagem quanto ao tempo de computação, pelo fato de não ser uma heurística construtiva, apresentando semelhanças com metaheurísticas como *Simulated Annealing* e Busca Tabu.

Rajendran e Ziegler (1997) propuseram a heurística RZ, que consiste de duas fases: na primeira, uma seqüência inicial é gerada utilizando uma regra de prioridade similar ao *shortest weighted total processing time*; na segunda fase, a seqüência inicial obtida é melhorada por meio de inserções das tarefas em seqüências parciais, sucessivamente obtidas.

Wang *et al*. (1997) desenvolveram duas heurísticas denominadas LIT (*less idle time*) e SPD (*smallest process distance*). Suas heurísticas não são comparadas com as existentes, mas sim com um limitante inferior do tempo médio de fluxo, proposto por Ahmadi e Bagchi (1990).

Woo e Yim (1998) desenvolveram uma heurística denominada WY, comparando seu desempenho com a heurística de Rajendran (1993), e também com adaptações do NEH (NAWAZ *et al*., 1983) e CDS (CAMPBELL e*t al*., 1970) para o critério de minimização do tempo médio de fluxo. Eles verificaram que seu método apresentava desempenho superior ao dos outros, principalmente comparado com Rajendran (1993).

Liu e Reeves (2001) apresentaram uma heurística flexível construtiva $H(x)$, que produzia x programas $(x = 1, 2,.., n)$. O procedimento heurístico construía uma seqüência solução S adicionando tarefas uma de cada vez de um conjunto U de tarefas não programas. As tarefas em U eram classificadas de acordo com uma função de índice que dependia do número de tarefas já seqüenciadas. A função de índice era uma combinação ponderada entre o tempo total de máquina parada e o tempo total de fluxo.

Allahverdi e Aldowaisan (2002) apresentaram sete heurísticas denominadas $IHx \ (x = 1,2,..,7)$. O melhor desempenho foi alcançado pela heurística $IH7$, que é constituída de três fases. Na primeira fase, uma solução inicial é obtida aplicando-se a heurística WY (WOO e YIM, 1998). Esta primeira solução é utilizada na segunda fase como seqüência inicial para a heurística RZ (RAJENDRAN e ZIEGLER, 1997). Finalmente, a segunda solução é melhorada por meio de uma busca local, com procedimentos de permutação nas posições das tarefas.

Framinan e Leisten (2003) propuseram o método denominado $FL - IH7$, inicialmente denominado $IH7 - proposed$, que consiste em uma extensão do $IH7$ (ALLAHVERDI e ALDOWAISAN, 2002). A diferença relevante situa-se na obtenção da solução inicial do processo de três fases do $IH7$. No $FL - IH7$ tal solução inicial é obtida por um método construtivo que utiliza inicialmente o mesmo procedimento de obtenção de

seqüências parciais do NEH (NAWAZ *et al*., 1983), porém, ordenando as tarefas de acordo com a soma dos tempos de processamento não-decrescentes. A seguir, as seqüências parciais são melhoradas por um procedimento de busca completa nas respectivas Vizinhanças de Permutação. Por meio de uma experimentação computacional constatou-se que o $FL - IH7$ apresenta um desempenho melhor quando comparado com $IH7$ original.

Framinan *et al*. (2005) apresentaram uma extensão da pesquisa realizada por Framinan e Leisten (2003). Na pesquisa as heurísticas *proposed* e $IH7 - proposed$ são respectivamente renomeadas para FL e $IH7 - FL$. Os autores apresentam duas novas heurísticas compostas, nomeadas de $C1$ e $C2$. As novas heurísticas $C1$ e $C2$ são comparadas com as heurísticas FL e $IH7 - FL$ e também com a original heurística $IH7$ proposta por Allahverdi e Aldowaisan (2002). Os resultados da experimentação computacional mostraram que as heurísticas existentes apresentam desempenho inferior que a heurística $C2$.

Li *et al*. (2006) propuseram uma nova heurística com a aceleração do cálculo do tempo total de fluxo, onde uma nova seqüência gerada é resultante da composição de duas subseqüências. Três heurísticas compostas são apresentadas ($ICH1$, $ICH2$ e $ICH3$), e os resultados computacionais mostraram que a heurística $ICH3$ apresentou os melhores resultados e também eficiência computacional quando comparada a outros métodos existentes na literatura.

Mais recentemente, Nagano e Moccellin (2007) apresentaram um novo método heurístico denominado $NM - flowtime$. A heurística proposta é composta de três fases. Na primeira as tarefas são ordenadas de acordo com os valores não-decrescentes das somas dos tempos de processamento para cada tarefa em todas as máquinas. A segunda fase utiliza-se o método de inserção de tarefas semelhante ao método NEH (NAWAZ *et al*., 1983) para obtenção da solução inicial. Finalmente, a solução inicial é melhorada utilizando procedimentos de busca local baseados em vizinhanças de permutação e inserção de tarefas para as parciais seqüências obtidas.

Em relação aos métodos melhorativos Rajendran e Ziegler (2004) desenvolveram dois métodos metaheurísticos baseado no algoritmo de otimização de colônia de formigas (*Ant-colony algorithm* (Stützle, 1998)). O primeiro método estende a idéia do algoritmo de colônia de formigas proposto por Stützle (1998) chamado de $MMAS$ (*max-min ant system*), incorporando a sugestão proposta por Merkle e Middendorf (2000) e uma nova proposta de busca local ($M - MMAS$). O segundo método é chamado de $PACO$ (*proposed ant-colony algorithm*). Os métodos foram avaliados com os 90 problemas (*benchmark*) de Taillard (1993). Os resultados obtidos mostraram que os dois métodos de colônia de formigas obtiveram melhores resultados que o $MMAS$, em média, para o critério de minimização do *makespan*. Considerando o critério de minimização da soma total dos tempos de fluxos das tarefas, em relação as melhores soluções dos problemas de Taillard (1993) apresentados por Liu e Reeves (2001), os algoritmos de colônia de formigas denominados de $M - MMAS$ e $PACO$ obtiveram melhores resultados, mostrando serem claramente superiores em comparação às heurísticas de Liu e Reeves (2001), e também em relação às melhores soluções reportadas por Liu e Reeves (2001).

Recentemente, Tasgetiren *et al.*, (2007) apresentaram uma metaheurística PSO (*swarm optimization algorithm*) para os mesmos problemas apresentados por Rajendran e Ziegler (2004). Em sua pesquisa uma regra heurística chamada SPV (*smallest position value*) proposta por Bean (1994) foi desenvolvida para habilitar o PSO para aplicação aos problemas de sequencimento. Adicionalmente, uma eficiente busca local, chamada de VNS (*variable neighborhood search*) é incorporada ao PSO. O método PSO foi testado para os problemas (*benchmark*) de Taillard (1993) e também para os problemas (*benchmark*) Watson *et al*. (2002). Para o critério de minimização da soma total dos tempos de fluxos verifica-se que as soluções obtidas são melhores comparadas a Liu e Reeves (2001) e Rajendran e Ziegler (2004).

De acordo com a revisão da literatura efetuada na pesquisa relatada neste trabalho, o método PSO_{VNS} proposto por Tasgetiren *et al*., (2007) é o melhor método metaheurístico em termos de qualidade da solução para a programação *flow shop* permutacional com o critério de minimização da soma total dos tempos de fluxo da tarefa.

3. *Clustering Search* (*CS*)

A metaheurística *Clustering Search* (CS), proposta por Oliveira e Lorena (2004, 2007), consiste de um processo de agrupamentos de soluções para detectar regiões supostamente promissoras no espaço de busca. O objetivo de encontrar tais regiões assim que possível é adaptar a estratégia de busca sobre elas. Uma região pode ser vista como um sub-espaço de busca definido por uma relação de vizinhança.

No CS um processo de agrupamento iterativo é executado simultaneamente com uma metaheurística, identificando grupos de soluções que merecem especial interesse. As regiões destes grupos de soluções devem ser exploradas tão logo sejam detectadas, através de heurísticas específicas de busca local. Espera-se uma melhoria no

processo de convergência associado a uma diminuição no esforço computacional em virtude do emprego mais racional dos métodos de busca local.

O *CS* procura localizar regiões promissoras através da representação destas regiões por *clusters*. Um *cluster* é definido pela tripla $G = \{C, r, \beta\}$ onde C, r e β são, respectivamente, o centro, raio de uma região de busca, e uma estratégia de busca associada ao *cluster*.

O centro C é uma solução que representa o *cluster*, identificando a sua localização dentro do espaço de busca, e pode ser alterado durante o processo. Inicialmente, os centros são obtidos aleatoriamente e progressivamente tendem a deslocar-se para pontos realmente promissores no espaço de busca. O raio r estabelece a distância máxima, a partir do centro, até a qual uma solução pode ser associada ao *cluster*. Por exemplo, r pode ser definido como o número de movimentos necessários para transformar uma solução em outra. A estratégia de busca β é uma sistemática de intensificação de busca, na qual soluções de um *cluster* interagem entre si, ao longo do processo de agrupamento, gerando novas soluções.

O *CS* consiste em quatro componentes conceitualmente independentes com diferentes atribuições: uma metaheurística (*ME*), um agrupador iterativo (*AI*), um analisador de agrupamentos (*AA*) e um algoritmo de otimização local (*AO*). A Figura 1 ilustra os quatros componentes que interagem entre si, o conjunto de soluções e os centros dos *clusters*.

O componente *ME* trabalha como um gerador iterativo de soluções de tempo integral. O algoritmo é executado independentemente dos componentes restantes e precisa ser capaz de gerar soluções de forma contínua para o processo de agrupamento. Simultaneamente, *clusters* são mantidos para representar estas soluções. Este processo trabalha como um laço infinito, no qual, soluções são geradas ao longo das iterações.

O componente *AI* objetiva reunir soluções similares dentro de *clusters*, mantendo uma solução no centro do *cluster* que seja representativa para as demais soluções. Para evitar um esforço computacional extra, *AI* é desenvolvido como um processo *online*, no qual os agrupamentos são progressivamente alimentados por soluções geradas em cada iteração de *ME*. Um número máximo de *clusters MC* é definido a priori, evitando a criação ilimitada de *clusters*. Uma métrica de distância também precisa ser definida inicialmente, permitindo uma medida de similaridade para o processo de agrupamento. Cada solução recebida por *AI* é agrupada no *cluster* que for mais similar a ela, causando uma perturbação em seu centro. Tal perturbação é chamada de assimilação e consiste basicamente em atualizar o centro com a nova solução recebida.

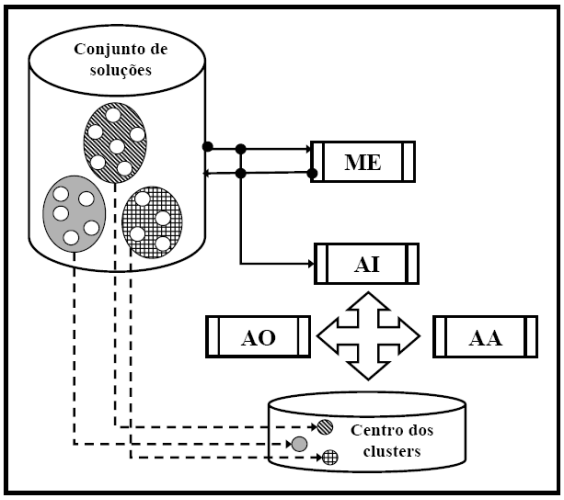

Figura 1. Diagrama Conceitual do CS

O componente *AA* provê uma análise de cada *cluster*, em intervalos regulares, indicando um provável *cluster* promissor. A densidade do *cluster*, λ_i, é a medida que indica o nível de atividade dentro do *cluster*. Por simplicidade, λ_i pode ser o número de soluções geradas por *ME* e alocadas para o *cluster* i. Sempre que λ_i atinja certo limitante, significando que algum modelo de informação foi predominantemente gerado por *ME*, tal *cluster* precisa ser mais bem investigado para acelerar o processo de convergência deste. *AA* também é responsável por eliminar os *clusters* com baixa densidade, permitindo criar novos *clusters* e preservando os *clusters* mais ativos. A eliminação do *cluster* não afeta o conjunto de soluções de *ME*, somente o centro do *cluster* é considerado relevante para o processo.

Por fim, o componente *AO* é um módulo de busca local que provê a exploração de uma suposta região

promissora, delimitada pelo *cluster*. Este processo acontece após *AA* ter descoberto um *cluster* promissor e uma busca local é aplicada no centro do *cluster*. *AO* pode ser considerado como a estratégia de busca β associada com o *cluster*, ou seja, uma heurística específica para o problema, que será empregada dentro do *cluster*.

4. *Evolutionary Clustering Search* para o Problema de Flow Shop Permutacional

Neste trabalho foi utilizada uma técnica denominada *Evolutionary Clustering Search* (*ECS*) que combina Algoritmos Genéticos (*AG*) e *Clustering Search* (*CS*), aplicado ao problema de programação *Flow Shop* Permutacional. No *ECS* o componente *ME* da metaheurística *CS* é implementado com um *AG* para geração de soluções que permitirão a exploração de regiões promissoras pelos demais componentes de *CS*.

No algoritmo *ECS* empregado neste trabalho foram feitas diversas modificações na concepção geral do *CS* apresentada anteriormente. O algoritmo *ECS* está ilustrado de forma geral na figura 2.

Como a qualidade dos indivíduos da população inicial é de grande importância no resultado do *AG*, para garantir esta qualidade, no processo de inicialização da população foi utilizada a variação da heurística de Nawaz *et al.* (1983), conhecida como *NEH*. Inicialmente um conjunto de *n* tarefas é ordenado de acordo com os valores não-decrescentes da soma dos tempos de processamento em todas as máquinas, em seguida, as duas primeiras tarefas da ordenação são seqüenciadas de modo a diminuir a soma do tempo total de fluxo desta seqüência parcial, as demais tarefas, a partir da terceira, são então inseridas (uma a uma) em uma posição da seqüência parcial que leve à menor soma total do tempo de fluxo das tarefas.

```
Procedimento ECS()
{
  Inicialize a população P;
  Inicialize o conjunto de clusters C;
  Enquanto (critério_Parada = Falso) {
    Enquanto (i < novos_Individuos) {
      pai1 ← Selecione entre 40% melhores de P;
      pai2 ← Selecione individuo qualquer em P;
      filho ← Recombinação(pai1, pai2);
      BuscaLocal(filho) com 60% de probabilidade;
      Se (Insere_na_população(filho))
        Componente_AI(filho, C);
      i ← i + 1;
    }
    Para cada cluster c ∈ C
      Se ( Componente_AA(c))
        Componente_AO(c);
  }
}
```

Figura 2 - Algoritmo *ECS*.

A representação usada no *AG* para os cromossomos foi um vetor com *n* posições, cada uma delas com uma tarefa, indicando diretamente a seqüência de tarefas da solução. O tamanho da população do *AG* foi definido em 500 indivíduos para que houvesse espaço na população inicial tanto para soluções aleatórias que contribuem para diversidade, quanto para soluções de maior qualidade geradas por uma versão modificada do *NEH*.

O primeiro indivíduo inserido na população inicial foi gerado pelo procedimento *NEH*. Outros indivíduos foram gerados por uma modificação do *NEH* em que, após a ordenação, as duas primeiras tarefas a serem seqüenciadas são definidas aleatoriamente entre as *n* tarefas do problema, o que resulta em *n(n-1)/2* possibilidades.

Com o objetivo de garantir tanto qualidade quanto diversidade na população inicial, o número de indivíduos gerados pelo *NEH* modificado foi dado por:

$$Mínimo\left\{\frac{n(n-1)}{4}, \frac{500}{2}\right\}$$

Assim, a menos do primeiro indivíduo gerado pelo *NEH* original, a população inicial teve apenas metade do seu número de indivíduos, ou metade dos indivíduos possíveis de serem produzidos pelo *NEH* modificado, o que for menor, como indivíduos de maior qualidade. Os indivíduos restantes foram gerados com permutações aleatórias das *n* tarefas, garantindo alto grau de diversidade.

A avaliação dos indivíduos foi feita diretamente pelo critério de minimização da soma do tempo total de fluxo das tarefas, e a rotina de inserção de indivíduos manteve a população ordenada, com o melhor indivíduo (aquele com a menor soma total do tempo de fluxo) ocupando a primeira posição na população. Esta mesma rotina de inserção também não permitiu a inserção de indivíduos repetidos na população.

Uma rotina de inicialização de *clusters* foi criada para aproveitar os bons indivíduos da população inicial. Esta rotina varria a população no sentido do melhor para o pior indivíduo, gerando novos *clusters* ou assimilando os indivíduos nos *clusters* já criados. Novos *clusters* foram criados quando o indivíduo em questão não se encontrava dentro do raio $r = 0{,}85 \cdot n$ de nenhum dos *clusters* já existentes. A assimilação de um indivíduo era feita no *cluster* de cujo centro este indivíduo se encontrava mais próximo. A medida de distância entre as permutações de indivíduos p_i e os centros dos *clusters* c_j foi adotada como sendo a quantidade de trocas necessárias para transformar p_i em c_j. O processo de geração desses *clusters* iniciais terminava quando toda a população era varrida ou quando era gerado o número máximo de *clusters*, neste trabalho determinado como 200. Tanto o raio de proximidade para assimilação quanto o número máximo de clusters foram escolhidos após uma série de testes preliminares. Foram procurados parâmetros que permitissem um comportamento do processo evolutivo em que, após um início com vários clusters sendo criados, o número de clusters fosse lentamente sendo reduzido pela ação do componente *AA* do *Cluster Search*, prevalecendo clusters nas regiões do espaço com as melhores soluções.

O processo de assimilação de um indivíduo em um *cluster* teve como base o processo *Path Relinking* (Glover, 1996). Partindo de um indivíduo p_i, sucessivas trocas de um par de genes eram feitas até que seu cromossomo se tornasse idêntico ao cromossomo do centro do cluster. A cada troca um novo cromossomo era gerado e avaliado. O par de genes escolhido para troca era aquele que produziria o cromossomo com melhor avaliação a cada passo. Ao final, se o próprio indivíduo p_i sendo assimilado, ou o melhor cromossomo encontrado nas sucessivas trocas, tivesse melhor avaliação que o centro do *cluster*, este passaria a ser o novo centro deste *cluster*.

Como critério de parada do processo evolutivo foi adotado o número máximo de 100 iterações ou o número de 20 iterações sucessivas sem que nenhum novo indivíduo fosse inserido na população. O número de tentativas de inserção de novos indivíduos a cada iteração foi igual a 50 (10% do tamanho da população).

A cada tentativa de gerar um novo indivíduo dois pais eram selecionados na população, um deles entre os melhores 40% da população, chamado de indivíduo base, e outro entre todos da população, chamado de guia. Um processo de cruzamento, ou recombinação, conhecido *BOX* (*Block Order Crossover*) (Syswerda, 1989), foi então usado para gerar os novos indivíduos. Nesta técnica, os indivíduos pais são combinados através da cópia aleatória de blocos de genes de ambos os pais, o que resulta na geração de um único filho contendo a carga genética dos pais. Neste trabalho, o filho foi gerado com 50% dos genes de cada pai. A figura 3 ilustra o esquema de recombinação *BOX*.

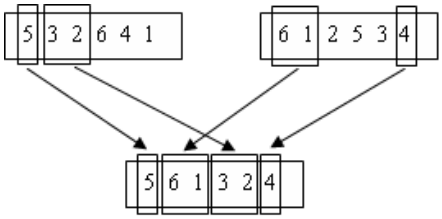

Figura 3 – Recombinação BOX.

Após a recombinação, o novo indivíduo tinha 60% de probabilidade de passar por um processo de melhoria na forma de uma busca local, conforme apresentada na figura 4.

Esta busca por uma solução melhorada foi feita por um processo híbrido que utiliza dois tipos de vizinhança (de permutação e de inserção). Na vizinhança de permutação todos os possíveis pares de genes do cromossomo são trocados, gerando assim $n \cdot (n-1)/2$ novos cromossomos. Na vizinhança de inserção, cada gene é removido de sua posição e inserido em todas as possíveis posições entre os demais genes, deslocando-os lateralmente para preencher a posição deixada por ele, gerando assim $n \cdot (n-1)$ novos cromossomos. A figura 4 ilustra o algoritmo de busca local (*LS1*).

Cada novo indivíduo, melhorado pela busca local na maioria das vezes, e que não fosse idêntico a nenhum indivíduo já pertencente à população, era inserido na população em uma posição referente à sua avaliação, provocando assim a remoção do pior indivíduo presente na população até aquele momento. Portanto, o processo evolutivo eventualmente atualizava a população a cada novo indivíduo gerado.

Os novos indivíduos inseridos com sucesso na população passavam pelo componente *AI* da metaheurística *CS*. Neste processo, ou os indivíduos são assimilados pelo *cluster* de cujo centro estão mais próximos, ou geram novos *clusters*. Os testes mostraram que nas primeiras quatro ou cinco iterações do processo evolutivo o número de *clusters* aumentava, mas em seguida declinava em função da eliminação dos *clusters* sem assimilações, aqueles referentes a regiões possivelmente não promissoras.

```
Busca_LS1(Solução_inicial)
{
  sa ← Solução_inicial;  // solução atual
  Fim ← Falso;
  Enquanto (Fim = Falso) {
    vp ← Vizinhança_permutação(sa);
    sp ← Varre vp até que sp seja melhor que sa
         ou sp seja o melhor em vp;
    vi ← Vizinhança_inserção(sa);
    si ← Varre vi até que si seja melhor que sa
         ou si seja o melhor em vi;
    Se (sp é melhor que si e sa)
      sa ← sp;
    Senão
      Se (si é melhor que sp e sa)
        sa ← si;
      Senão
        Fim ← Verdadeiro;
  }
  Retorna(sa);
}
```

Figura 4 – Busca local de melhoria dos indivíduos.

Após a geração de 50 novos indivíduos, cada um dos *clusters* era analisado pelo componente *AA* da metaheurística. Neste processo, os clusters que tivessem passado por alguma assimilação na iteração eram considerados promissores e passavam pelo processo equivalente ao componente *AO*, implementado com uma segunda busca local (*LS2*) ilustrada na figura 5. Os *clusters* que não tivessem passado por nenhuma assimilação nas últimas 5 iterações eram eliminados.

```
Busca_LS2(Solução_inicial)
{
  sa ← Solução_inicial; // solução atual
  Fim ← Falso
  Enquanto (Fim = Falso) {
    vi ← Vizinhança_inserção(sa);
    si ← Varre Vi até que si seja melhor que sa
         ou si seja o melhor em vi;
    Se (si é melhor que sa) {
      sa ← si;
      vp ← Vizinhança_permutação(sa);
      sp ← Varre vp até que sp seja melhor que sa
           ou sp seja o melhor em vp;
      Se (sp é melhor que sa)
        sa ← sp;
    } Senão {
      vp ← Vizinhança_permutação(sa);
      sp ← Varre Vp até que sp seja melhor que sa
           ou sp seja o melhor em vp;
      Se (sp é melhor que sa)
        sa ← sp;
      Senão
        Fim ← Verdadeiro;
    }
  }
  Retorna(sa);
}
```

Figura 5 – Busca local do componente AO de *CS* .

Durante todo o processo, o melhor *cluster*, isto é, aquele cujo centro tinha a melhor avaliação, era mantido salvo. No final das iterações, o primeiro indivíduo da população e o centro do melhor *cluster* eram as soluções obtidas, respectivamente, pelo componente *ME* (gerador de soluções) e pela evolução dos *clusters*.

5. Experimentação Computacional e Métodos de Análise

Para a avaliação do desempenho do método proposto, denominado *ECS - FS* (*Evolutionary Clustering Search-Flow Shop*), foram realizadas experimentações computacionais utilizando os problemas de Taillard (1993) conforme utilizados por Liu e Reeves (2001), Rajendran e Ziegler (2004), Li *et al*. (2006) e Tasgetiren *et al*. (2007).

Neste trabalho, o *ECS - FS* foi codificado em linguagem *C* e processado em um microcomputador Pentium IV de 3.0 GHz, com 1 GByte de RAM.

As estatísticas usadas para avaliar o desempenho do método foram a Porcentagem de Sucesso e o Desvio Relativo Médio. A primeira é definida pelo quociente entre o número total de problemas para os quais o método obteve a melhor soma total dos tempos de fluxo das tarefas, e o número total de problemas resolvidos. A segunda quantifica o desvio relativo (DR_h) que o método h obtém em relação a melhor soma dos tempos de fluxo das tarefas obtido para um mesmo problema, sendo calculado conforme segue:

$$DR_h = \frac{(F_h - F_*)}{F_*}$$

onde, F_h : soma dos tempos de fluxo das tarefas obtido pelo método h ;
F_* : melhor soma dos tempos de fluxo das tarefas obtido pelos métodos, para um determinado problema.

6. Análise dos Resultados

Para avaliação do desempenho, os resultados do método *ECS-FS* foram comparados com dois algoritmos baseados em colônias de formigas (*M-MMAS* e *PACO*), apresentados por Rajendran e Ziegler (2004), e um método baseado em enxame de partículas (*PSO*) apresentado por Tasgetiren *et al*. (2007). Os resultados principais são apresentados nas Tabelas 1 e 2.

A tabela 1 apresenta a melhor solução obtida com os métodos. Verifica-se assim a superioridade do novo método *ECS-FS* para o conjunto de problemas avaliados. Para o total de 90 problemas o *ECS - FS* apresentou a melhor solução em 82 problemas o que corresponde a 91,1% do total de problemas, sendo que 59 soluções são inéditas (65,5%).

A tabela 2 apresenta a porcentagem de sucesso por classes de problemas. Pode-se verificar que o método *ECS - FS* apresentou porcentagens de sucesso entre 70% e 100%. A tabela 2 mostra ainda que a diferença entre os desvios relativos médios do método *ECS - FS* e os desvios dos outros métodos é bem acentuada, consubstanciando a superioridade do método proposto, já mostrada pelas porcentagens de sucesso.

Em relação ao tempo médio de computação, o método *ECS-FS* apresentou um intervalo de variação de 8,62 a 27548,11 segundos entre as classes dos menores (20x5) e dos os maiores (100x20) problemas.

Tabela 1 – Melhores soluções obtidas para os problemas de Taillard

n	*m*	M-MMAS	PACO	PSO_{VNS}	ECS-FS
20	5	14056	14056	**14033**	**14033**
		15151	15214	**15151**	**15151**
		13416	13403	**13301**	**13301**
		15486	15505	**15447**	**15447**
		13529	**13529**	**13529**	**13529**
		13139	**13123**	**13123**	**13123**
		13559	13674	**13548**	**13548**
		13968	14042	**13948**	**13948**
		14317	14383	**14295**	**14295**
		12968	13021	**12943**	**12943**
20	10	20980	20958	**20911**	**20911**
		22440	22591	**22440**	**22440**
		19833	19968	**19833**	**19833**
		18724	18769	**18710**	**18710**
		18644	18749	**18641**	**18641**
		19245	**19245**	19249	**19245**
		18376	18377	**18363**	**18363**
		20241	20377	**20241**	**20241**
		20330	**20330**	**20330**	**20330**
		21320	21323	**21320**	**21320**
20	20	**33623**	**33623**	34975	**33623**
		31604	31597	32659	**31587**
		33920	34130	34594	**33920**
		31698	31753	32716	**31661**
		34593	34642	35455	**34557**
		32637	32594	33530	**32564**
		33038	**32922**	33733	**32922**
		32444	32533	33008	**32412**
		33625	33623	34446	**33600**
		32317	32317	33281	**32262**
50	5	65768	65546	65058	**64838**
		68828	68485	68298	**68159**
		64166	64149	63577	**63453**
		69113	69359	68571	**68310**
		70331	70154	69698	**69477**
		67563	67664	67138	**66902**
		67014	66600	**66338**	66355
		64863	65123	64638	**64471**
		63735	63483	63227	**63068**
		70256	69831	69195	**69092**
50	10	89599	88942	88031	**87683**
		83612	84549	83624	**83535**
		81655	81338	80609	**80365**
		87924	88014	87053	**86934**
		88826	87801	87263	**86865**
		88394	88269	87255	**86969**
		90686	89984	**89259**	89304
		88595	88281	**87192**	87316
		86975	86995	**86102**	86213
		89470	89238	88631	**88534**
50	20	127348	126962	128622	**126315**
		121208	121098	122173	**119502**
		118051	117524	118719	**116910**
		123061	122807	123028	**121031**
		119920	119221	121202	**118914**
		122369	122262	123217	**121087**
		125609	125351	125586	**123340**
		124543	124374	125714	**123005**
		124059	123646	124932	**122203**
		126582	125767	126311	**124785**
100	5	257025	257886	**254762**	254911
		246612	246326	245315	**243943**
		240537	241271	239777	**239002**
		230480	230376	**228872**	228888
		243013	243457	242245	**241659**
		236225	236409	**234082**	234172
		243935	243854	242122	**241753**
		234813	234579	232755	**232315**
		252384	253325	249959	**249608**
		246261	246750	244275	**244210**
100	10	305004	305376	303142	**301176**
		279094	278921	277109	**276902**
		297177	294239	292465	**290844**
		306994	306739	304676	**304377**
		290493	289676	288242	**287545**
		276449	275932	272790	**272635**
		286545	284846	282440	**282381**
		297454	297400	**293572**	294119
		309664	307043	305605	**304964**
		296869	297182	295173	**294362**
100	20	373756	372630	374351	**371391**
		383614	381124	379792	**376383**
		380112	379135	378174	**374599**
		380201	380765	380899	**378550**
		377268	379064	376187	**374426**
		381510	380464	379248	**377567**
		381963	382015	380912	**378367**
		393617	393075	392315	**389680**
		385478	380359	382212	**380152**
		387948	388060	386013	**383928**

Tabela 2 – Porcentagem de sucesso e Desvio relativo médio (%)

n	*m*	M-MMAS	PACO	PSO_{VNS}	ECS-FS
20	5	20[a]	20	**100**	**100**
		0,1975[b]	0,4544	**0,0000**	**0,0000**
20	10	60	20	90	**100**
		0,0492	0,3235	0,0021	**0,0000**
20	20	20	20	0	**100**
		0,1195	0,1892	2,8278	**0,0000**
50	5	0	0	10	**90**
		1,1302	0,9450	0,2452	**0,0026**
50	10	0	0	30	**70**
		1,4196	1,1569	0,1841	**0,0322**
50	20	0	0	0	**100**
		1,2852	0,9780	1,8421	**0,0000**
100	5	0	0	30	**70**
		0,8733	0,9921	0,1638	**0,0104**
100	10	0	0	10	**90**
		1,2714	0,9834	0,2189	**0,0186**
100	20	0	0	0	**100**
		1,0678	0,8361	0,6627	**0,0000**

[a]Porcentagem de sucesso;
[b]Desvio relativo médio.

7. Considerações Finais

Os resultados experimentais mostraram que o método heurístico híbrido *ECS-FS* tem um desempenho superior tanto na porcentagem de sucesso quanto no desvio relativo médio, em comparação com os melhores resultados referenciados na literatura, utilizando os problemas de Taillard (1993) para solução do problema de programação da produção em ambientes *flow shop*, e tendo como medida de desempenho a redução do estoque em processamento (minimização do soma total dos tempos de fluxos das tarefas). Além de obter soluções de melhor qualidade, o esforço computacional é aceitável para fins práticos.

O problema clássico de seqüenciamento de tarefas em um ambiente de produção *flow shop* tem sido objeto de intensos esforços de pesquisa nos últimos 50 anos e, para fins práticos, tal problema pode ser considerado já resolvido. Entretanto, tendo em vista sua complexidade, ainda permanece como uma direção de pesquisa a busca de métodos heurísticos e metaheurísticos cada vez mais eficazes quanto à qualidade da solução.

A realização da pesquisa relatada neste trabalho foi motivada pelas considerações acima, procurando resgatar as características essenciais de um método metaheuristico, ou seja, adequado equilíbrio entre a qualidade da solução e a eficiência computacional, simplicidade e facilidade de implementação.

Referências

Ahmadi, R. H. e Bagchi, U. (1990), Improved lower bounds for minimizing the sum of completion times of *n* jobs over m machines, *European Journal of Operational Research*, 44, 331-336.

Allahverdi, A. e Aldowaisan, T. (2002), New heuristics to minimize total completion time in *m*-machine flowshops, *International Journal of Production Economics,*77, 71-83.

Bean, J. C. (1994) Genetic algorithm and random keys for sequencing and optimization, *ORSA Journal on Computing*, 6, 154-160.

Campbell, H. G., Dudek, R. A. e Smith, M. L. (1970), A heuristic algorithm for *n*-job, *m*-machine sequencing problem, *Management Science,* 16, 630-637.

Framinan J. M., Leisten, R. e Ruiz-Usano R. (2005), Comparison of heuristics for flowtime minimisation in permutation flowshops, *Computer and Operations Research*, 32, 1237-1254.

Framinan, J. M. e Leisten, R. (2003), An efficient constructive heuristic for flowtime minimisation in permutation flow shop, *OMEGA*, 31, 311-317.

Garey, M. R., Johnson, D. S. e Sethi, R. (1976), The Complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research*, 1, 117-129.

Glover, F. (1996), Tabu search and adaptive memory programing: Advances, applications and challenges, In: *Interfaces in Computer Science and Operations Research*. Kluwer, 1–75.

Gupta, J. N. D. (1972), Heuristic algorithms for multistage flowshop scheduling problem, *AIIE Transactions,* 4, 11-18.

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

Li, X., Wang, Q. e Wu, C. (2006), Efficient composite heuristics for total flowtime minimization in permutation flow shops, *Omega* (2006), doi: 10.1016/j.omega.2006.11.003.

Liu, J. e Reeves C. R. (2001), Constructive and composite heuristic solutions to the $P//\sum C_i$ scheduling problem. *European Journal of Operational Research*, 132, 439-452.

Merkle, D. e Middendorf, M. (2000), An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In: *Proceedings of the Evo Workshops*, In: *Lecture Notes in Computer Science*, 1803, Springer-Verlag, Berlin, 287-296.

Miyazaki, S., Nishiyama, N. e Hashimoto, F. (1978), An adjacent pairwise approach to the mean flowtime scheduling problem, *Journal of the Operations Research Society of Japan*, 21, 287-299.

Nagano, M. S., Moccellin, J. V. (2007), Reducing mean flow time in permutation flow shop. *Journal of the Operational Research Society* (2007), doi: 10.1057/palgrave.jors.2602395.

Nawaz, M., Enscore, E. E. e Ham, I. (1983), A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem, *OMEGA*, 11, 91-95.

Oliveira, A. C. M. e Lorena, L. A. N. (2004). Detecting promising areas by evolutionary clustering search. Em: Advances in Artificial Intelligence (Eds. Bazzan, A.L.C.; Labidi, S.). Springer Lecture Notes in Artificial Intelligence Series, 385–394.

Oliveira, A. C. M. e Lorena, L. A. N. (2007), Hybrid evolutionary algorithms and clustering search. Em: Springer SCI Series (Eds: Crina Grosan, Ajith Abraham and Hisao Ishibuchi), 2007 – aceito para publicação (http://www.lac.inpe.br/~lorena/alexandre/HEA-07.pdf).

Rajendran, C. (1993), Heuristic algorithm for scheduling in a flowshop to minimise total flowtime. *International Journal Production Economics*, 29, 65-73.

Rajendran, C. e Chaudhuri, D. (1991), An efficient heuristic approach to the scheduling of jobs in a flowshop. *European Journal of Operational Research*, 61, 318-325.

Rajendran, C. e Ziegler H. (2004), Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, *European Journal of Operational Research*, 155, 426-438.

Rajendran, C. e Ziegler, H. (1997), An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103, 129-138.

Rinnooy Kan, A. H. G. (1976), Machine Scheduling Problems: Classification, Complexity, and Computations. *The Hahue: Nijhoff*.

Stützle, T. (1998), Applying iterated local search to the permutation flowshop problem, Technical Report, AIDA-98-04, Darmstad University of Technology, Computer Science Department, Intelletics Group, Darmstad, Germany.

Syswerda, G. (1989), Uniform crossover in genetic algorithms. In: *International Conference on Genetic Algorithms* (ICGA), Virginia, USA, 2–9.

Taillard, E. (1993), Benchmarks for basic scheduling problems, *European Journal of Operational Research*, 64, 278-285.

Tasgetiren, M. F., Liang, Y. C., Sevkli, M. e Gencyilmaz, G. (2007), A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, *European Journal of Operational Research*, 177, 1930-1947.

Wang, C., Chu, C. e Proth, J. M. (1997), Heuristic approaches for $n/m/F/\Sigma C_i$ scheduling problems. *European Journal of Operational Research,* 96, 636-644.

Watson, J. P., Barbulescu, L., Whitley L. D. e Howe, A. E. (2002), Contrasting structured and random permutation flowshop scheduling problems: Search space topology and algorithm performance, *ORSA Journal of Computing*, 14, 98-123.

Woo, D. S. e Yim H. S. (1998), A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers & Operations Research*, 25, 175-182.