Heurística Evolutiva para a Minimização da Duração Total da Programação em Sistemas de Produção *No-Idle Flow Shop*

João Carlos Soriano Sampaio Januário
Universidade de São Paulo, Escola de Engenharia de São Carlos
Av. Trabalhador São-Carlense, 400 – CEP: 13566-590, São Carlos – SP
joao.januario@usp.br

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

RESUMO

Este artigo trata do problema de programação de operações em um ambiente de produção *no-idle flow shop*, tendo como objetivo a minimização da duração total da programação. Uma nova metaheuristica híbrida é proposta para a solução do problema. O método proposto é comparado com o melhor método reportado 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*, *no-idle*, duração total da programação, heurística evolutiva, *cluster search*.

ABSTRACT

This paper deals with no-idle flow shop scheduling problem with the objective of minimizing makespan. A new hybrid metaheuristic is proposed for the scheduling problem solution. The proposed method is compared with the best method reported in the literature. Experimental results show that the new method provides better solutions regarding the solution quality to set of problems evaluated.

Keywords: *flow shop, no-idle, makespan, evolutionary heuristic, cluster search.*

1. Introdução

O problema de programação de operações *flow shop* é um problema de programação da produção no qual *n* tarefas devem ser programadas, na mesma seqüência, em um conjunto de *m* máquinas distintas. Um caso específico de programação *flow shop*, denominando permutacional, é quando em cada máquina mantém-se a mesma ordem de processamento das tarefas.

A solução do problema consiste em determinar dentre as (*n*!) seqüências possíveis das tarefas, aquela que otimiza alguma medida de desempenho da programação, sendo que as mais usuais consistem na minimização da Duração Total da Programação (*makespan*), ou a minimização da soma dos tempos de Fluxo das tarefas (*total flowtime*). A primeira relaciona-se na utilização eficiente dos recursos (máquinas) enquanto que a segunda busca minimizar o estoque em processamento.

As restrições comumente consideradas no problema de programação *flow shop* restringem, mas não excluem as suas aplicações práticas na realidade (Dudek, Panwalkar e Smith (1992)). Uma das restrições é a de que as máquinas que compõem o *flow shop* não sofram interrupção uma vez que ela foi inicializada.

Este tipo de problema é conhecido como *no-idle flow shop* (*NIFS*), e ocorre quando o tempo de preparação ou o custo de utilização da máquina são relativamente elevados, considerando que desligá-la ou prepará-la mais vezes que o necessário provoca um processo bastante oneroso. Em todos os casos, a melhor alternativa é determinar um seqüenciamento que considera a restrição de não ociosidade nas máquinas. Este mesmo problema pode ser denotado por $F/prmu, no-idle/C_{max}$, e sua complexidade computacional foi considerada como do tipo *NP-hard* por Tanaev *et al*. (1994).

Por todos os motivos apresentados, verifica-se a importância e as dificuldades encontradas para a solução do problema *NIFS*. Este artigo tem por objetivo apresentar um novo método heurístico evolutivo denominando HE-*NIFS*, sendo este comparado com o mais recente método proposto da atualidade.

2. Métodos Heurísticos para o Problema *NIFS*

Adiri e Pohoryles (1982) foram os primeiros a tratar do problema *no-idle.* Neste trabalho, os autores também trataram do problema *no-wait flow shop* (*NWFS*). A principal contribuição foi o desenvolvimento de um algoritmo que soluciona otimamente o problema *F2/prmu,no-idle/*$\sum C_j$.

Os métodos heurísticos para o problema de *m*-máquinas foram inicialmente tratados por Woollam (1986) para o objetivo de minimização do *makespan*. Basicamente, vários métodos foram selecionados na literatura, incluindo o conhecido método NEH.

Narain e Bagga (2003) estudaram o problema *F3/prmu,no-idle/*$C_{máx}$. Eles apresentaram um método de programação linear inteira e um algoritmo *B & B* com resultados computacionais limitados. O mesmo problema com três máquinas foi estudado por Saadani *et al*. (2003). Eles propuseram um limitante inferior (*lower bound*) e um eficiente método heurístico. A heurística foi comparada com um método anterior apresentado em Saadani *et al.* (2001) e apresentaram melhores resultados.

Kamburowski (2004) propôs uma representação em rede que proporcionou uma melhor representação do problema identificando alguns paradoxos resultantes da condição de *no-idle*, e conduzindo algumas relações de dominância entre as máquinas em que o problema torna-se eficientemente resolvível.

Saadani *et al.* (2005) apresentou uma heurística baseada em *Travelling Salesman Problem* (*TSP*) para *F/prmu,no-idle/*$C_{máx}$. Basicamente, os autores modelaram como distâncias os *makespans* de pares de tarefas. A partir da distância mínima, a heurística aplica mecanismos de busca na vizinhança, inserindo as tarefas, uma por uma, em todas as posições possíveis. A heurística possui uma complexidade $O(n^3)$ e é facilmente implementável.

Kalczynski e Kamburowski (2005) propuseram uma heurística para o *F/prmu,no-idle/*$C_{máx}$ com a complexidade computacional de $O(n^2m)$. A heurística foi comparada com a de Saadani *et al.* (2005), e apresentou os melhores resultados na maior parte dos problemas

avaliados. Os autores apresentaram também uma adaptação da heurística NEH para o problema *no-idle*, e os resultados apresentaram que o método proposto supera também essa heurística.

Mais recentemente, o problema *NIFS* foi novamente estudado por Kalczynski e Kamburowski (2007), onde foram apresentadas situações especiais entre os problemas *no-wait* e *no-idle*. Os autores apresentaram uma representação em redes em que os comprimentos do caminho mais longo na rede representam os *makespans*. As redes revelaram a dualidade entre os dois problemas, e uma explicação gráfica do fato sobre a condição *no-wait* e *no-idle* para *makespan* foi discutida.

Baraz e Mosheiov (2008) propuseram um método heurístico de duas fases para $F/prmu,no-idle/C_{máx}$. Os autores concluíram que o tempo de execução da sua heurística é $O(n^2)$. No entanto, deve-se destacar que os autores não consideram a complexidade adicional do cálculo do *makespan* em cada etapa. Uma vez que este cálculo tem uma complexidade computacional $O(nm)$, eles demonstraram a superioridade da sua heurística proposta em comparação com os resultados obtidos por Saadani *et al.* (2005), entretanto não apresentaram uma comparação com a heurística de Kalczynski e Kamburowski (2005).

Pan e Wang (2008a, b) em dois trabalhos similares, propuseram um procedimento que acelera a busca na vizinhança de inserção e reduz a complexidade computacional de $O(n^3m)$ para $O(n^2m)$. Esta aceleração baseia-se em procedimentos apresentados em Taillard (1990) para a mesma busca na vizinhança, mas para o problema *flow shop* tradicional. Ambos os algoritmos utilizaram um método de busca chamado *Iterated Greedy* (Ruiz e Stützle (2007)). Os autores utilizam os problemas de Taillard (1993) do *flow shop* clássico para o problema *no-idle*. Nos dois trabalhos, foram comparados os métodos propostos com as heurísticas de Baraz e Mosheiov (2008) e Kalczynski e Kamburowski (2005). Os resultados indicam a superioridade dos métodos propostos, porém os resultados não foram comparados entre eles diretamente.

Finalizando, Uma extensiva experimentação computacional foi realizada por Ruiz *et al.* (2008) para o problema *no-idle flow shop*. Os autores realizaram duas experimentações computacionais: A primeira foi a avaliação de nove métodos heurísticos determinísticos adaptados ao problema *NIFS*; e a segunda foi a avaliação de mais quatro métodos metaheurísticos também adaptados ao problema *NIFS*.

Os resultados obtidos pelos autores na primeira experimentação computacional apresentaram que os métodos FRB3 e GH_BM2 foram os que apresentaram os melhores desempenhos. O primeiro com relação a ao desvio relativo médio, e o segundo com relação a qualidade dos resultados e tempo de computação.

Os resultados obtidos na segunda experimentação verificaram que o IG_{LS} apresentou resultados significativamente melhores que o HDPSO.

Finalizando de acordo com a revisão da literatura efetuada na pesquisa relatada neste trabalho, pode-se concluir que o método IG adaptado, bem como o proposto método melhorado GH_BM2, juntos com as heurísticas de Rad *et al.* (2009) constituem os melhores métodos existentes para o problema *NIFS* com critério de minimização do *makespan*.

3. Nova Heurística Evolutiva

A heurística evolutiva proposta nesta pesquisa é uma hibridização do *Cluster Search* (*CS*) apresentado por Ribeiro Filho, Nagano e Lorena (2007) com o método IG_{LS} proposto por Ruiz e Stützle (2007).

Neste processo evolutivo o critério de parada é determinado pelo tempo de computação do problema (*elapsed time*). O tempo limite será adotado como sendo $n(m/2)T$ milissegundos, onde T é um parâmetro regulador do critério de parada. Com um critério de parada proposto dessa maneira o tempo computacional será proporcional ao número de tarefas e máquinas. Assim, quanto maior o problema (n; m), maior o tempo disponível para as iterações.

A qualidade da solução obtida pelo *CS* é em grande parte dependente dos processos de inicialização presentes em sua estrutura. Entre essas inicializações existe a inicialização da população inicial e a inicialização dos *clusters*.

A população inicial é formada por um vetor de *k* posições, onde cada posição é uma solução. O tamanho desse vetor é definido como sendo o número de indivíduos possíveis de se gerar em um tempo limite, sendo 500 o valor máximo para *k*. Definindo esse tempo limite para geração da população como sendo 10% do tempo de computação do problema (*elapsed time)* já definido, a população inicial será de no máximo 500 indivíduos ou um valor limitado pelos 10% do *elapsed time*. Dessa maneira fica certo que no máximo 10% do tempo de computação do problema será gasto para se inicializar a população.

Para garantir a qualidade dos indivíduos gerados na população, é proposto uma combinação da heurística NEH de Nawaz *et al*. (1983) com o método *Iterated Greedy* (*IG*) (Ruiz e Stützle (2007)), no qual são geradas soluções (seqüências) através de duas fases: destruição e construção. Na fase de destruição alguns elementos da seqüência são removidos para que posteriormente sejam inseridos, reconstruindo a seqüência. A motivação para se utilizar uma combinação de métodos na heurística evolutiva proposta (HE-*NIFS*), foi a busca de uma maior variedade de soluções de melhor qualidade no processo de inicialização.

Na inicialização da população o primeiro indivíduo inserido na população é gerado pelo procedimento NEH. Os demais indivíduos são gerados pelo método *Iterated Greedy* (*IG*), aplicando as fases de destruição e construção a um indivíduo já criado. Assim, o primeiro indivíduo é gerado pelo procedimento NEH, o segundo será gerado pela destruição e posterior construção desse primeiro indivíduo. O terceiro será encontrado do processo de destruição e construção do segundo, e assim sucessivamente. Seguindo esse processo, a formação de um indivíduo necessita de um outro já formado, ou seja, o processo é recursivo, a menos do primeiro indivíduo que é gerado pela heurística NEH.

Tal procedimento visa proporcionar uma maior diversidade das soluções no processo de inicialização, sem que tenha que recorrer a um processo simplesmente aleatório, o que levaria a perda da qualidade das soluções geradas na população inicial.

A avaliação dos indivíduos gerados é feita diretamente pelo critério de otimização do *makespan*, e ela é a responsável por manter a população ordenada, com o melhor indivíduo (aquele com o menor *makespan*) ocupando a primeira posição. Este mesmo procedimento não permite a inserção de indivíduos repetidos na população.

Um procedimento de inicialização de *clusters* é criado para aproveitar os bons indivíduos da população inicial. Da mesma forma que a população, os *clusters* criados serão submetidos ao processo de avaliação para ordená-los. Terminado a sua inicialização, os *clusters* avaliados como tendo seus *makespans* entre os 1/3 melhores entre todos os gerados irão passar por um processo de busca local (LS1) tendo em vista uma melhor seleção do espaço de soluções a ser trabalhado pelo método *Cluster Search*.

A inicialização dos *clusters* varre a população no sentido do melhor para o pior indivíduo, gerando novos *clusters* ou assimilando os indivíduos nos *clusters* já criados. Novos *clusters* são criados quando o indivíduo em questão não se encontra dentro do raio (r=0,85n) de nenhum dos *clusters* já existentes, onde n é o número de tarefas da seqüência. A assimilação de um indivíduo é feita no *cluster* em que o centro este indivíduo se encontra 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 termina quando toda a população é varrida ou quando é gerado o número máximo de *clusters*, neste trabalho foi determinado como 200.

Uma característica do *Cluster Search* é o seu comportamento evolutivo em que, após um início com vários *clusters* criados, seu número é lentamente reduzido prevalecendo os *clusters* nas regiões do espaço com as melhores soluções.

O processo de assimilação de um indivíduo em um *cluster* tem como base o processo *Path Relinking* (Glover, 1996). Partindo de um indivíduo p_i, sucessivas trocas de um par de tarefas serão feitas até que sua seqüência se torne idêntica a seqüência do centro do *cluster*. A cada troca uma nova seqüência é gerada e avaliada. O par de tarefas escolhida para a troca é aquele que produz a seqüência com a melhor avaliação a cada passo. Ao final, se o próprio

indivíduo p_i assimilado, ou a melhor seqüência encontrada nas sucessivas trocas, tiver a melhor avaliação que o centro do *cluster*, este passa a ser o novo centro deste *cluster*.

A cada tentativa de gerar um novo indivíduo dois outros são selecionados na população, um deles entre os melhores 10% da população, chamado de indivíduo base, e outro entre todos da população, chamado de guia. Um processo de cruzamento, ou recombinação, conhecido como BOX (*Block Order Crossover*) proposto por Syswerda (1989), é 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 parte da herança dos pais. Neste trabalho, o filho foi gerado com 75% dos genes do pai 1 (base) e com 25% do pai 2 (guia).

Após a recombinação, o novo indivíduo tem 60% de probabilidade de passar por um processo de melhoria na forma de uma busca local. Essa melhoria pode ser feita através de dois tipos de busca local, LS1, com 40% de probabilidade de ocorrer, ou LS2, com 20%.

Tanto a LS1 como a LS2 é um processo híbrido que utiliza dois tipos de vizinhança, a permutação e a inserção. Ambas as buscas estão detalhadas nas Figuras 1 e 2, sendo a Figura 1 sobre a LS1 e a Figura 2 sobre a LS2. Outra característica relevante nestas buscas é o seu caráter aleatório, isso porque ambas são feitas através da escolha aleatória e sem repetição das tarefas a serem trabalhadas na vizinhança. Tal procedimento foi proposto, mais uma vez, para garantir uma maior diversidade de soluções.

Na vizinhança de permutação todos os possíveis pares de tarefas da seqüência são trocados, gerando assim $n(n-1)/2$ novas seqüências. Na vizinhança de inserção cada tarefa é removida de sua posição e inserida em todas as outras possíveis posições, deslocando-as lateralmente para preencher a posição deixada por ela, gerando assim $(n-1)^2$ novas seqüências.

Cada novo indivíduo, melhorado por uma das duas buscas locais na maioria das vezes, e que não é idêntico a nenhum indivíduo já pertencente à população, é inserido na população em uma posição referente à sua avaliação, provocando assim a remoção do pior indivíduo presente na população até o momento. Portanto, o processo evolutivo eventualmente atualiza a população a cada novo indivíduo gerado.

Os novos indivíduos inseridos com sucesso na população são assimilados pelo *cluster* de cujo centro estão mais próximos, ou geram novos *clusters*.

Buscando selecionar ainda mais o espaço de soluções a ser trabalhado pelo *Cluster Search* uma nova busca local (LS2) será efetuada nos *clusters* classificados como estando na porção dos 1/3 melhores pelo critério de avaliação de *makespan*. Essa segunda busca local será efetuada assim que o tempo de computação do método HE-*NIFS* for superior a 50% do tempo de computação proposto pelo critério de parada (*elapsed time)*. Essa busca local será efetuada uma única vez, não se repetindo até o final do método proposto.

Durante todo o processo o melhor *cluster*, isto é, aquele cujo centro tem a melhor avaliação, será mantido salvo. Sendo este a solução apresentada pelo *Cluster Search*.

Para uma melhor compreensão do *Cluster Search* a Figura 3 apresenta o procedimento detalhadamente.

```
Busca Local LS1(π)
    fim := 0;
    enquanto (fim = 0) e (tempo < 10% do Tempo total de processamento);
        sp ← melhor seqüência π' ou a própria sequência π após a aplicação da vizinhança de permutação
             na seqüência π;
        si ← melhor seqüência π' ou a própria seqüência π após a aplicação da vizinhança de inserção na
             seqüência π;
        se (Makespan (sp)< Makespan (si)) e (Makespan (sp)< Makespan (π));
             π ← sp;
        senão (Makespan (si)< Makespan (sp)) e (Makespan (si)< Makespan (π));
             π ← si;
        senão fim := 1;
```

Figura 1 – Busca Local LS1

```
Busca Local LS2(π)
    fim := 0;
    enquanto (fim = 0) e (tempo < 10% do Tempo total de processamento);
        si ← melhor seqüência π' ou a própria seqüência π após a aplicação da vizinhança de inserção na
             seqüência π;
        se Makespan (si)< Makespan (π);
            π ← si;
            sp ← melhor seqüência π' ou o próprio π após a aplicação da vizinhança de permutação
                 na seqüência π;
            se Makespan (sp)< Makespan (π);
                π ← sp;
        senão sp ← melhor seqüência π' ou o próprio π após a aplicação da vizinhança de permutação
                   em π;
            se Makespan (sp)< Makespan (π);
                π ← sp;
        senão fim := 1
```

Figura 2 – Busca Local LS2

```
Cluster Search(π)
    tls2 := 0
    enquanto tempo < 10% do Tempo total de processamento;
        para i ← 1 até n;
            P(i) ← indivíduo gerado na inicialização do população;
        Ordenação da população inicial
        para i ← 1 até n;
            P(i) pode gerar um novo cluster ou sofrer assimilação (Inicialização dos clusters);
    enquanto tempo < Tempo total de processamento;
        Criação de um novo indivíduo (Recombinação Box);
        Melhoramento do novo indivíduo em 60% dos casos;
        Assimilação ou criação de um novo cluster através do novo indivíduo;
        se (tempo > 50% do Tempo total de processamento) e (tls2 = 0);
            tls2 := 1;
            Os 1/3 melhores clusters passarão pela busca local LS2;
```

Figura 3 – Procedimento *Cluster Search*

4. Experimentação Computacional

Para a avaliação do desempenho do método HE-*NIFS* o método foi comparado com o melhor método reportado na literatura conhecido por IG_{LS} proposto Ruiz *et al*. (2008) (http://www.upv.es/deioac/Investigacion/Ruiz_Vallada_Fernandez.pdf). Todos os métodos foram codificados em linguagem C e processados em um microcomputador Intel Xeon 3GHz e 32GB de memória RAM.

Foi utilizado o banco de dados de Taillard composto por 10 problemas para cada classe, totalizando 120 problemas testes.

As estatísticas utilizadas para avaliar o desempenho dos métodos foram a Porcentagem de Sucesso (PS) e o Desvio Relativo (DR).

A primeira estatística é definida pelo quociente entre o número total de problemas para os quais o método obteve a melhor solução, e o número total de problemas resolvidos. A segunda quantifica o desvio relativo (DR_h) que o método *h* obtém em relação a melhor solução para um mesmo problema, sendo calculado por:

$$DR_h = \frac{(M_h - M_*)}{M_*}$$

Onde:

M_h : *makespan* obtido pelo método *h*; e M_* : melhor *makespan* obtido pelos métodos para um determinado problema.

5. Análise dos Resultados

Os métodos IG_{LS} e HE-*NIFS* foram avaliados considerando o critério de parada *n(m/2)T* para $T \in \{50, 250, 500, 750, 1000\}$.

A Figura 4 apresenta a porcentagem de sucesso dos métodos IG_{LS} e HE-*NIFS* para o total de problemas avaliados de acordo com a variação do parâmetro *T*.

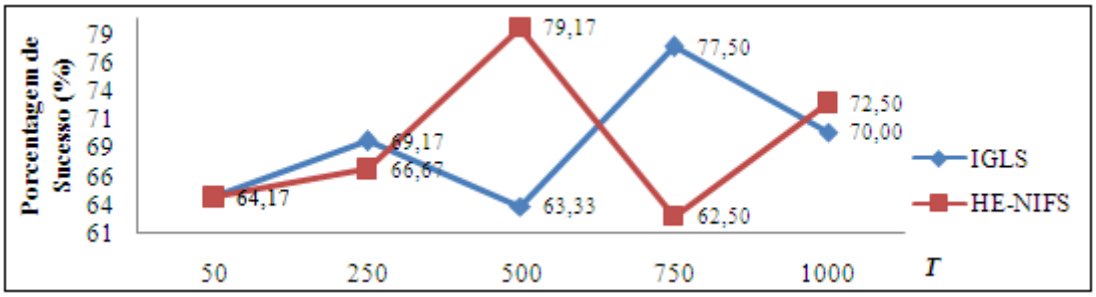

Figura 4 – Porcentagem de sucesso dos métodos IG_{LS} e HE-*NIFS*

A Figura 5 apresenta a significância estatística da diferença entre as heurísticas IG_{LS} e HE-*NIFS* para os valores de *T* referente aos desvios relativos. Nela foi levantada a média dos pares das heurísticas correspondente a 95% dos intervalos de confiança.

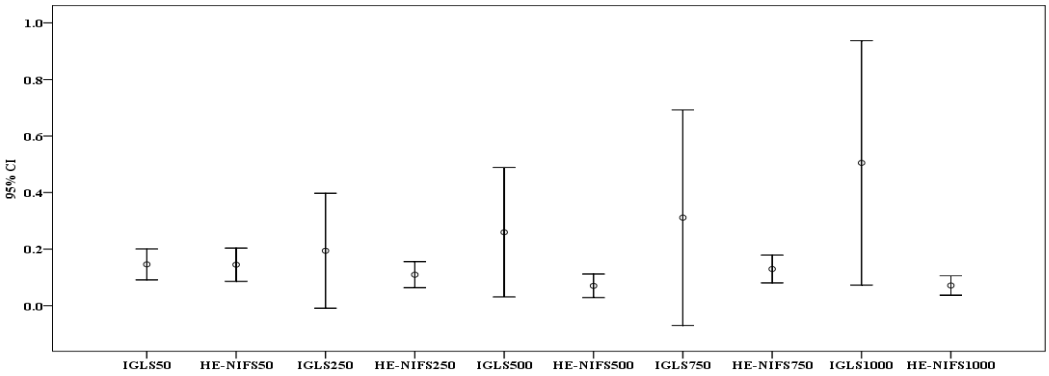

Figura 5 – Média e intervalos de confiança (95%) para pares de algoritmos avaliados

Pode-se observar na Figura 6 que a qualidade da solução entre os métodos IG_{LS} e HE-*NIFS* apresentam diferenças significativas a medida que os valores de *T* são avaliados. O método HE-*NIFS* apresenta-se mais estável com relação a qualidade das soluções obtidas, por outro lado o IG_{LS} apresenta um aumento crescente da variação do intervalo de confiança implicando em soluções de menor qualidade conforme o aumento crescente do valor de *T*.

Os resultados prévios da experimentação computacional apresentados neste trabalho são motivadores, apesar da porcentagem de sucesso obtido pelo método HE-*NIFS* não ser muito superior em comparação ao método IG_{LS}, os resultados indicam que o método HE-*NIFS* obtém soluções de melhor qualidade apresentando pouca variação quando comparado a IG_{LS}.

Nesse sentido verifica-se assim a superioridade do novo método HE-*NIFS* em relação a obtenção de soluções de melhor qualidade comparando o conjunto de problemas avaliados.

6. Considerações Finais

Os resultados experimentais mostraram que o método heurístico HE-*NIFS* apresentou desempenho igual relacionando a porcentagem de sucesso, e superior em relação ao desvio relativo (qualidade da soluções obtidas) em comparação com o método IG_{LS}, que é considerado o melhor método da atualidade. Assim, com tais resultados apresentados pode-se afirmar que o método HE-*NIFS* é um método alternativo de altíssima qualidade para o problema de programação da produção em ambientes *no-idle flow shop* com critério de minimização do *makespan*.

Apesar dos resultados iniciais apresentados já serem considerados satisfatórios, o método HE-*NIFS* pode ainda ser melhorado encontrando os parâmetros ótimos, que poderão ser encontrados através de uma exaustiva experimentação computacional, onde poderão ser avaliadas as variações dos parâmetros e suas respectivas soluções fornecidas. Este processo constitui uma segunda parte deste trabalho que terá como objetivo a divulgação em periódicos especializados.

Referências Bibliográficas

Adiri, I. and Pohoryles, D. (1982). *Flowshop no-idle or no-wait scheduling to minimize the sum of completion times*. Naval Research Logistics, 29(3):495–504.

Baraz, D. and Mosheiov, G. (2008). *A note on a greedy heuristic for flow-shop makespan minimization with no machine idle-time*. European Journal of Operational Research, 184(2):810–813.

Dudek, R. A., Panwalkar, S. S., Smith, M. L. (1992). *The lessons of flowshop scheduling research*. Operations Research. vol. 40, 7-13.

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

Kalczynski, P. J. and Kamburowski, J. (2005). *A heuristic for minimizing the makespan in no-idle permutation flow shops*. Computers & Industrial Engineering, 49(1):146–154.

Kalczynski, P. J. and Kamburowski, J. (2007). *On no-wait and no-idle flow shops with makespan criterion*. European Journal of Operational Research, 178(3):677–685.

Kamburowski, J. (2004). *More on three-machine no-idle flow shops*. Computers & Industrial Engineering, 46(3):461–466.

Narain, L. and Bagga, P. C. (2003). *Minimizing total elapsed time subject to zero total idle time of machines in n × 3 flowshop problem*. Indian Journal of Pure & Applied Mathematics, 34(2):219–228.

Nawaz, M., Enscore, Jr, E. E., and Ham, I. (1983). *A heuristic algorithm for the m-machine, n-Job flow-shop sequencing problem*. OMEGA, The International Journal of Management Science, 11(1):91–95.

Pan, Q.-K. and Wang, L. (2008a). *No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm*. In press at International Journal of Advanced Manufacturing Technology.

Pan, Q.-K. and Wang, L. (2008b). *A novel differential evolution algorithm for noidle permutation flow-shop scheduling problems*. European Journal of Industrial Engineering, 2(3):279–297.

Rad, S. F., Ruiz, R., and Boroojerdian, N. (2009). *New high performing heuristics for minimizing makespan in permutation flowshops*. OMEGA, the International Journal of Management Science, 37:331–345.

Ribeiro Filho, G., Nagano, M. S., Lorena, L. A. N. (2007). *Evolutionary Clustering Search for Flowtime Minimization in Permutation Flow Shop*. T. Bartz-Beielstein et al. (Eds.): HM 2007, LNCS-Lecture Notes in Computer Science, vol. 4771, 69–81.

Ruiz, R. and Stützle, T. (2007). *A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem*. European Journal of Operational Research, 177(3):2033–2049.

Ruiz, R., Vallada, E. y Fernández-Martínez, C. (2008). *Scheduling in flowshops with no-idle machines*. Informe técnico DEIOAC-2008-1. Departamento de Estadística e Investigación Operativa Aplicadas y Calidad. Universidad Politécnica de Valencia.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2001). *A travelling salesman approach to solve the F/no − idle/Cmax problem*. In Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'01, volume 2, pages 880–888, Quebec, Canada.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2003). *Three stage no-idle flow-shops*. Computers & Industrial Engineering, 44(3):425–434.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2005). *A travelling salesman approach to solve the F/no−idle/Cmax problem*. European Journal of Operational Research, 161(1):11–20.

Taillard, E. (1990). *Some efficient heuristic methods for the flow shop sequencing problem*. European Journal of Operational Research, 47:67–74.

Tanaev, V. S., Sotskov, Y. N., and Strusevich, V. A. (1994). *Scheduling Theory. Multi-Stage Systems*. Kluwer Academic Publishers, Dordrecht.

Woollam, C. R. (1986). *Flowshop with no idle machine time allowed*. Computers & Industrial Engineering, 10(1):69–76.