Algoritmo baseado em Estratégias Evolutivas para Minimização do Makespan em um Problema de Sequenciamento de Tarefas Flowshop Híbrido e Flexível

Eduardo C. de Siqueira
Centro Federal de Educação Tecnológica de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil
Email: eduardos@dppg.cefetmg.br

Marcone J. F. Souza
Departamento de Computação, Campus Universitário
Universidade Federal de Ouro Preto
Ouro Preto, Minas Gerais, Brasil
Email: marcone@iceb.ufop.br

Sérgio R. de Souza
Centro Federal de Educação Tecnológica de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil
Email: sergio@dppg.cefetmg.br

***Resumo*—Este trabalho propõe um algoritmo baseado em Estratégias Evolutivas para resolução do problema de sequenciamento *flowshop* híbrido e flexível. A construção das soluções iniciais é feita por dois métodos heurísticos: a heurística NEH aleatória e a metaheurística IG. O principal mecanismo de busca no espaço de soluções dos algoritmos desenvolvidos é a mutação, que é feita com base em dois parâmetros, sendo um relativo à probabilidade de aplicar cada tipo de mutação, e outro relativo ao número de vezes em que cada mutação é aplicada. Feita a mutação, uma parcela da população de filhos é refinada por um procedimento de busca local baseado no método Iterative Improvement Insertion (III). Ao final das gerações é aplicado um método de busca local com representação completa nos indivíduos. O algoritmo foi testado e comparado com um algoritmo da literatura e os resultados mostraram que ele é capaz de gerar soluções melhores para um subconjunto de instâncias do problema.**

I. Introduction

Problemas de sequenciamento aparecem em vários setores de atividade, entre eles o da produção industrial. Nesse contexto, o problema consiste em definir uma sequência de tarefas a serem executadas em um conjunto de máquinas, satisfazendo a um conjunto de restrições operacionais e atendendo a um certo objetivo, que pode ser a minimização do tempo de término de todas as tarefas, a minimização dos tempos de atraso na conclusão das tarefas, entre outros.

Existem diversos estudos a respeito dos problemas de sequenciamento, porém sempre existiu uma distância entre teoria e prática. Muitos modelos desenvolvidos não incorporam várias restrições operacionais, limitando, assim, sua aplicação em situações reais. Há, no entanto, uma tendência recente em desenvolver abordagens de solução para problemas reais [1].

Nesse sentido, este trabalho considera o problema real descrito em [1], que trata uma variação do problema de *flowshop* híbrido (HFS, do inglês *Hybrid Flowshop*), em que um conjunto de tarefas passa por um conjunto de estágios e para cada estágio existe um conjunto de máquinas paralelas não-relacionadas. A característica de um *flowshop* é que o fluxo de processamento nas máquinas é o mesmo, ou seja, todas as tarefas seguem a mesma sequência de estágios. No entanto, no problema considerado os estágios podem não ser todos executados. Tal variante, denominada *Flowline* Híbrido e Flexível (HFFL, do inglês *Hybrid and Flexible Flowline*), é, desta forma, uma generalização do HFS e do *Flowline* Flexível. O critério de otimização considerado é o de minimizar o maior tempo de conclusão das máquinas, o chamado *makespan*.

Tendo em vista o fato de o HFFL ser da classe NP-difícil [1], este trabalho propõe um algoritmo baseado em Estratégias Evolutivas (ES, do termo em inglês *Evolutionary Strategies*) [2], [3] para sua solução. Segundo [3], as ES foram desenvolvidas para aplicação em problemas de otimização numérica, e se caracterizam pela sua robustez e eficiência. As ES são procedimentos iterativos que realizam a busca pelo espaço de soluções através de populações de indivíduos que representam potenciais soluções para o problema de otimização. Cada iteração é, por analogia com os sistemas biológicos, chamada de geração. Em cada geração, os operadores genéticos de recombinação e mutação atuam sobre a população, permitindo a exploração do espaço de busca. O princípio da sobrevivência do mais apto é implementado pelo mecanismo de seleção, que garante a sobrevivência dos indivíduos que representam soluções de melhor qualidade para o problema. O uso dessa classe de algoritmos foi motivado pela sua aplicação bem sucedida na solução de vários problemas de otimização, como os relatados em [4] e [5]. A adaptação proposta para a ES é inspirada no trabalho de [4] e de [6], esse último trabalho apresenta um algoritmo baseado em Diferencial Evolution com

Iterated Greedy.

O restante deste trabalho está organizado da seguinte forma. Na seção II é feito um levantamento bibliográfico dos problemas de *flowshop*. Na seção III o problema é formulado e exemplificado. O algoritmo proposto é descrito na seção IV. Na seção V são mostrados os resultados encontrados, enquanto a última seção conclui o trabalho e aponta os trabalhos futuros.

II. Literature review

Como comentado anteriormente, o HFFL é uma generalização do *flowshop* híbrido (HFS). O HFS, por sua vez, é um problema diferente do *Flowshop* Flexível e do *Flowline* Flexível, pois para esses dois últimos problemas as máquinas disponíveis em cada estágio são idênticas. O HFS não tem essa restrição. Alguns estágios podem ter apenas uma máquina, mas pelo menos um estágio deve ter um grupo de máquinas em paralelo. Estas máquinas geralmente são diferentes [7].

Segundo [8], os trabalhos de pesquisa sobre agendamento HFS apareceram na década de 1970. Um dos primeiros trabalhos sobre HFS na modelagem do sistema de produção em uma indústria de fibras sintéticas é o de [9]. Em [10] é mostrado que o problema HFS com o objetivo *makespan* é NP-completo. Assim, como o HFFL é uma generalização do HFS, então o HFFL também o é.

Em [11] é feito um levantamento sobre as aplicações de computação evolucionária a problemas do mundo real em indústrias de metais, de papel e química. Em [12] é propostae a resolução de um problema de escalonamento em um *flowshop* realístico com tempos de *setup* dependentes da sequência e diferentes pesos relativos para cada tarefa. O objetivo, nesse caso, é a minimização do atraso máximo ponderado e do atraso ponderado total.

No trabalho [13] é feita uma revisão sobre algoritmos exatos e heurísticas para resolução multiobjetivo de problemas de sequenciamento em ambientes *flowshop*. Entre os algoritmos abordados estão *branch-and-bound*, Busca Tabu, *Simulated Annealing*, Colônia de Formigas, Algoritmos Genéticos etc.

Várias heurísticas para resolução de um *flowshop* com várias máquinas idênticas em cada estágio, restrições de precedência, tempos de *setup* não dependentes da sequência, entre outras características, são propostas em [14]. Em [15] são desenvolvidas variações de heurísticas para resolver um problema da *flowline* flexível com máquinas paralelas idênticas, tempos de *setup* dependentes da sequência e com o objetivo de minimizar o *makespan*.

Em [16] são apresentados seis diferentes modelos matemáticos para resolução de uma generalização do problema *flowshop* de permutação. O objetivo a ser otimizado é a minimização do *makespan*. Os modelos foram testados com instâncias de 4 até 16 tarefas e nenhum deles foi capaz de alcançar a solução ótima em tempo de processamento aceitável para todas as instâncias. Então, foram propostos 14 metodos heurísticos para resolução desse problema.

Em [17], os autores propõem um modelo matemático e a aplicação de heurísticas construtivas para um problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas com tempos de *setup* dependentes da sequência e dos recursos atribuídos. Tem-se como objetivos a minimização do *makespan* e a minimização da combinação linear do total dos recursos atribuídos. Com este modelo foi possível alcançar a melhor solução possível do problema em instâncias de até 8 tarefas e 5 máquinas.

Em [18] são propostos algoritmos genéticos para um problema HFS com máquinas paralelas não relacionadas para cada estágio, tempos de *setup* dependentes da sequência e de elegibilidade de máquina. Os resultados alcançados nesse trabalho mostraram que o algoritmo proposto foi mais eficiente do que heurísitcas como *Simulated Annealing*, Colônia de Formigas entre outras.

Em [19] é feita uma revisão de algoritmos exatos, heurísticas e meta-heurísticas para o HFS. Um método de relaxação lagrangiana com geração de cortes para esse problema, com o objetivo de minimizar o somatório dos atrasos ponderados é apresentado em [20]. Em [21] é utilizado um algoritmo baseado em uma estratégia de agregação e separação em um problema de HFS, com o objetivo de minimizar o *makespan* e aumentar a capacidade produtiva das máquinas. No trabalho [22] é proposto um algoritmo de seleção clonal para resolução de um problema HFS. Nesse caso, o objetivo é a minimização do somatório das penalizações por atraso e por adiantamento.

Em [23] é apresentado uma nova modelagem matemática para o HFS, enquanto que em [24] é modelado um problema de *Flowshop* Híbrido em linhas de montagem de placas de circuito. Nesses dois trabalhos foram utilizados Algoritmos Genéticos, com o objetivo de minimizar o *makespan* nos problemas propostos.

Em [25] é proposta a aplicação de heurísticas construtivas para um problema de *flowshop* híbrido com dois estágios e recirculação. No trabalho [26] os autores também propõem a aplicação de heurísticas para o problema de *flowshop* híbrido com dois estágios; nesse caso, existem máquinas paralelas não-relacionadas no segundo estágio. O objetivo, nos dois trabalhos, é a minimização do *makespan*.

Em [27] é proposto um algoritmo heurístico para resolução de um problema *flowshop* híbrido com elegibilidade de máquinas. Os objetivos a serem otimizados são a minimização do somatório do *completion time* dos grupos de tarefas e a minimização da soma das diferenças entre o tempo de conclusão e o tempo de entrega desses grupos. Em [28] utilizam um algoritmo baseado em *Particle Swarm Optimization* (PSO) para a resolução desse problema.

Um modelo matemático para um novo problema de sequenciamento *flowshop* híbrido e flexível, denominado *Flowline* Híbrido e Flexível (HFFL, do inglês *Hybrid and Flexible Flowline*) é apresentado em [1]. Com este modelo foi possível resolver na otimalidade o HFFL em problemas testes de até 15 tarefas, 3 estágios e 2 máquinas em cada estágio. Nesse trabalho também foram apresentadas algumas heurísticas de construção para resolução de problemas de maior porte, de 50 e 100 tarefas.

No trabalho [29] é chamada a atenção para duas questões importantes a respeito de HFFL: a determinação da sequência em cada estágio e a distribuição das tarefas nas máquinas em cada estágio. É apresentado um algoritmo baseado na meta-heurística *Iterated Local Search* (ILS) com o objetivo de minimizar o *makespan*.

Em [30] é desenvolvido um processo de solução baseado em Algoritmo Genético para o HFFL. O algoritmo foi implementado em plataformas de computação sequenciais e paralelas, e os desempenhos avaliados e comparados. Nos trabalhos [31] e [32] também propostos Algoritmos Genéticos para resolução desse problema. O objetivo nesses dois últimos trabalhos é a minimização do *makespan*.

III. Problem formulation

Seja um conjunto de tarefas $N = \{1, 2, 3, ..., n\}$, um conjunto de estágios $M = \{1, 2, 3, \cdots, m\}$ pelos quais cada tarefa pode passar e um conjunto M_i de máquinas paralelas não-relacionadas para processar essas tarefas. No HFFL tem-se as seguintes características:

1) F_j: Conjunto de estágios que a tarefa j visita, sendo $1 < F_j < m$;
2) p_{ilj}: Tempo de processamento da tarefa j na máquina l e estágio i.
3) rm_{il}: Tempo de *release* da máquina l no estágio i. Ele representa o tempo de início dos processos na máquina. Nenhuma tarefa pode iniciar na máquina antes do tempo de *release*.
4) E_{ij}: Conjunto de máquinas elegíveis para a tarefa j no estágio i.
5) lag_{ilj}: Tempo de atraso entre o fim da tarefa j, na máquina l do estágio i, e o início do próximo estágio da tarefa j.
6) S_{iljk}: Tempo de preparação (*setup*) da máquina l no estágio i, quando a tarefa k é executada logo após a tarefa j. Existe um valor binário associado, A_{iljk}, que indica se o *setup* é antecipativo, ou seja, A_{iljk} assume o valor 1 se a preparação pode ser feita antes que a tarefa seja liberada na fase anterior e o valor 0, caso contrário.

Considera-se o objetivo de minimizar o *makespan*, ou seja, o tempo de término da última tarefa do sistema.

Para exemplificar o problema, seja uma instância com cinco tarefas e dois estágios, três máquinas em cada estágio. A Tabela I mostra quais máquinas são elegíveis para cada tarefa em cada estágio. Nesta Tabela, j indica uma tarefa, e a linha i representa cada um dos 3 estágios. Por essa tabela, pode-se verificar, por exemplo, que a tarefa 1 pode ser executada nas máquinas 2 e 3, no estágio 1, e na máquina 4, no estágio 2. Pode-se verificar também, que as tarefas 1, 2 e 5 visitam todos os estágios, enquanto as tarefas 3 e 4 pulam os estágios 1 e 2, respectivamente.

A Tabela II contém o tempo de *release* para cada máquina, a linha rm_{il} indica os tempos de *release* para cada máquina, e cada célula p_{ilj} os tempos de processamento. Por exemplo, o tempo de release da máquina 1 é igual a 20 e o da máquina 2 é igual 5.

Tabela I
ELEGIBILIDADE

	i	1	2
j	1	{2,3}	{4}
	2	{3}	{4,5,6}
	3	-	{5}
	4	{1,3}	-
	5	{1,2}	{4,5,6}

Tabela II
TEMPO DE *release*

i	1			2		
l	1	2	3	4	5	6
rm_{il}	20	5	33	38	29	45

A Tabela III tempo de processamento de cada tarefa em cada máquina, enquanto a Tabela IV mostra os tempos de atraso. Nestas duas tabelas, j indica uma tarefa, a linha i representa cada um dos 3 estágios, e a linha l representa cada uma das 5 máquinas. Nestas tabelas, pode-se verificar que o tempo de processamento da tarefa 1 na máquina 2 é de 8, e o tempo de atraso é nulo, enquanto a mesma tarefa na máquina 3 tem um tempo de processamento de 13 e tempo de atraso igual 9. O tempo de processamento de uma tarefa em uma máquina não elegível é nulo.

A Tabela V mostra os tempos de preparação (tempos de *setup*), que são dependentes da sequência, e seus valores binários associados. Esse valor binário representa se o *setup* é antecipativo (1), ou não (0). Nesta Tabela, j e k indicam uma tarefa, a linha i representa cada um dos 3 estágios, e os dados de cada máquina em cada estágio estão separados por vírgula. Por exemplo, o tempo de *setup* entre as tarefas 1 e 4 na máquina 2 do primeiro estágio é igual a 8 e o valor binário igual a 1 (antecipativo), porém o *setup* na máquina 1 não existe pois essa máquina não é elegível para a tarefa 4.

A Figura 1 ilustra um possível sequenciamento para esse exemplo. Note que o *makespan* para esse sequenciamento é igual a 63.

Tabela III
TEMPO DE PROCESSAMENTO

	i	1			2		
	l	1	2	3	4	5	6
j	1	0	8	13	21	0	0
	2	0	0	15	45	46	44
	3	0	0	0	0	13	0
	4	12	0	32	0	0	0
	5	18	16	0	16	19	36

Tabela IV
TEMPO DE ATRASO

	i	1		
	l	1	2	3
j	1	0	0	9
	2	0	0	8
	3	0	0	0
	4	2	0	-6
	5	0	-5	0

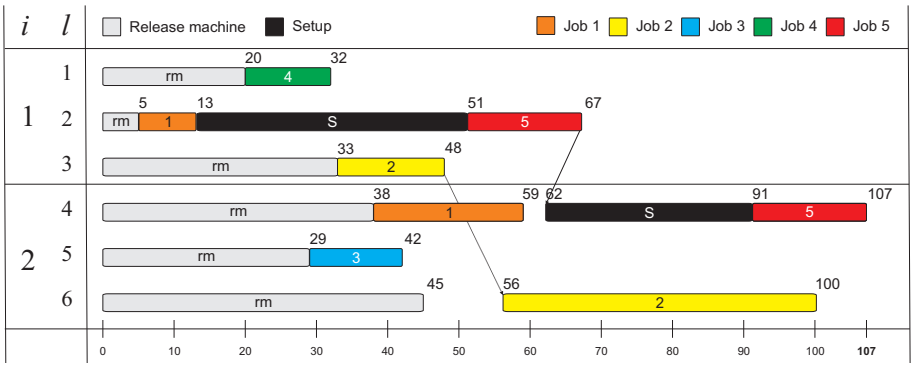

Figura 1. Diagrama de GANTT. $C_{max} = 107$

Tabela V
TEMPOS DE *setup* E VALORES A_{iljk}

	i	1				
	k	1	2	3	4	5
j	1	-,-,-	-,-,5(0)	-,-,-	-,-,10(0)	-,38(0),-
	2	-,-,10(0)	-,-,-	-,-,-	-,-,38(0)	-,-,-
	3	-,-,-	-,-,-	-,-,-	-,-,-	-,-,-
	4	-,-,46(0)	-,-,21(0)	-,-,-	-,-,-	43(0),-,-
	5	-,6(0),-	-,-,-	-,-,-	25(0),-,-	-,-,-
	i	2				
j	1	-,-,-	33(1),-,-	-,-,-	-,-,-	29(0),-,-
	2	30(1),-,-	-,-,-	-,6(1),-	-,-,-	8(1),39(1),27(1)
	3	-,-,-	-,45(1),-	-,-,-	-,-,-	-,30(0),-
	4	-,-,-	-,-,-	-,-,-	-,-,-	-,-,-
	5	41(0),-,-	47(0),7(0),38(1)	-,48(0),-	-,-,-	-,-,-

IV. METHODOLOGY

Nesta seção é apresentado um algoritmo baseado em Evolutionary Strategies (ES). Uma parcela da população inicial é gerada por meio de uma adaptação da heurística de construção NEH [33], e a outra parcela é construída com base na metaheurística *Iterated Greedy* – IG [34]. O algoritmo fazuso do procedimento *Iterative Improvement Insertion* – III [34] para refinar parte dos indivíduos gerados durante o processo evolutivo. Ao final das gerações é aplicada um método de busca local com representação completa nos indivíduos, proposto em [35].

Esta seção está organizada como segue. Na subseção IV-A é mostrado como um indivíduo é representado. Na subseção IV-B são mostradas as mutações usadas para explorar o espaço de soluções. Na subseção IV-C é detalhado o algoritmo ES básico, enquanto nas subseções IV-D e IV-E são apresentados os métodos NEH e IG para gerar soluções iniciais. A subseção IV-F traz a explicação do funcionamento do método de busca local com representação completa.

A. Solution Representation

Um indivíduo ind do problema é representado por uma dupla (π, M), em que π é a sequência de tarefas para processamento e M é uma matriz de máquinas, assim como por dois vetores de parâmetros de mutação ($prob$ e a). Nesta representação, todos os estágios contém a mesma sequência de tarefas (π), chamada representação permutacional, e para cada tarefa está associada uma coluna da matriz M identificando em qual máquina a tarefa é executada em cada estágio. Caso a tarefa não seja executada em dado estágio é atribuído o valor nulo. Os vetores $prob$ e a contêm um número de posições igual à quantidade de tipos diferentes de mutação, sendo que em cada posição de $prob$ armazena-se a probabilidade de aplicar um tipo de mutação, enquanto que em a armazena-se a intensidade dessa mutação.

A Figura 2 ilustra a dupla (π, M) para o indivíduo do exemplo considerado na Figura 1.

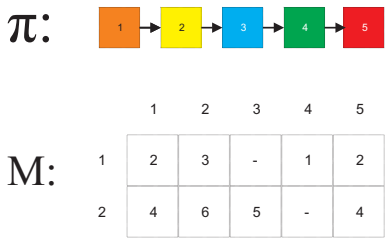

Figura 2. Representação de parte de um indivíduo

Na Figura 2 verifica-se, por exemplo, que a sequência de tarefas é 1, 2, 3, 4 e 5, ou seja, a tarefa 1 precede a 2, que por sua vez precede a 3 e a 4, finalmente, a tarefa 5 é a última. Verifica-se, também, pela matriz M, que a tarefa 1 é executada na máquina 2 no primeiro estágio, na máquina 4 no segundo estágio. O valor nulo constante na primeira linha e segunda coluna da matriz M indica que a tarefa e não é executada no primeiro estágio.

Para a busca local com representação completa cada estágio passa conter uma sequência de tarefas. Assim, nessa representação, para o exemplo mostrado, a sequência do estágio 1 é 1, 2, 4 e 5, e no estágio 2 é 1, 2, 3 e 5.

B. Mutation Description

A exploração do espaço de soluções do problema é feita com base em dois tipos de mutação:

- Realocação na sequência (MRS): consiste em escolher uma tarefa e mudá-la para uma nova posição na sequência;
- Realocação em Bloco na sequência (MBRS): consiste em escolher um bloco de tarefas adjacentes e mudá-lo para uma nova posição na sequência;

Neste trabalho são considerados blocos de 2, 3 e 4 tarefas. Assim, o MBRS se subdivide em três outros tipos de mutação, que se diferem pelo tamanho do bloco de tarefas.

C. Evolutionary Strategies

Como dito anteriormente, os dois algoritmos propostos têm como base a heurística Evolutionary Strategies (ES), cujo pseudocódigo está esquematizado no Algoritmo 1.

Na primeira fase do algoritmo ES (linhas 1 a 7 do Algoritmo 1), é gerada uma população inicial de μ indivíduos. Neste caso, metade dos indivíduos são gerados pelo método Greedy-Random NEH, e a outra metade são construídos com base na metaheurística IG.

A segunda fase (linhas 8 a 20) consiste em gerar uma população de γ filhos por meio de mutação. A seguir, é aplicado o método III para uma quantidade κ de filhos, com $\kappa < \gamma$. O III não é aplicado a todos os indivíduos da população de filhos em vista de seu alto custo computacional. Após esses procedimentos, são selecionados, dentre a população de pais e filhos, os indivíduos sobreviventes para a próxima geração. Esses passos são repetidos até que um critério de parada seja atendido. Cada módulo de ES é detalhado a seguir.

Nas linhas 3 a 5 do Algoritmo 1 é formado um indivíduo *ind* a partir da aplicação dos vetores *prob* e a a um indivíduo (sequência) s. Para tanto, o vetor de probabilidade de mutação *prob* de cada indivíduo é inicializado com um número real escolhido aleatoriamente no intervalo [0, 1] para cada um dos quatro tipos de mutação, enquanto que o vetor a de intensidade de mutação desse indivíduo é inicializado com um número inteiro escolhido aleatoriamente no conjunto $\{1, \cdots, 8\}$.

Na linha 12 do Algoritmo 1 os vetores de parâmetros de mutação são modificados. Para o vetor *prob* de probabilidade aplica-se uma mutação seguindo uma distribuição normal com desvio-padrão σ_{real} e média zero. Para o vetor a de intensidade aplica-se uma mutação seguindo uma distribuição binomial com desvio-padrão $\sigma_{binomial}$ e média zero.

A mutação dos indivíduos (linha 13 do Algoritmo 1) funciona como segue. Para cada tipo de mutação é gerado um número real aleatório entre 0 e 1, e verificado se esse valor satisfaz a condição de probabilidade $prob_l$. Caso afirmativo, retira-se a_l tarefas (ou bloco de tarefas) da solução corrente e as insere em um vetor π_R. Depois, para reconstruir a solução, cada tarefa (ou bloco de tarefas) de π_R é inserida na melhor posição possível do sequenciamento.

Na linha 17 do Algoritmo 1 é feito o refinamento de κ indivíduos filhos pelo método III. Este refinamento consiste em selecionar uma tarefa da solução corrente e inseri-la na melhor posição possível do sequenciamento. Se houver melhora na solução corrente, ela é atualizada, e o procedimento reiniciado. Esses passos são repetidos até que todas as tarefas sejam consideradas e não haja melhora na solução. A ordem de seleção das tarefas é aleatória.

Os μ indivíduos que comporão a próxima geração (linha 19 do Algoritmo 1) são aqueles com os menores valores de *makespan*, dentre a população de pais e filhos.

Após o critério de parada ser atingido, é aplicado o método de busca local com representação completa nos indivíduos da população sobrevivente. Essa representação é aqui considerada no intuito de explorar melhor o espaço de busca. Visto que a representação permutacional, embora alcance bons resultados, restringe o espaço de busca. Por fim, é retornado o melhor indivíduo da população final.

Algorithm 1 ES

Require: μ, γ, κ, $criterioParada$
Ensure: $MelhorIndividuo$
  for $w \leftarrow 1$ to μ **do**
    $\pi_w \leftarrow$ geraSolucao();
    Inicialize o vetor de probabilidade de mutação a_w;
    Inicialize o vetor de intensidade de mutação $prob_w$;
    Forme ind_w aplicando os vetores $prob_w$ e a_w a π_w;
    $Pop_w \leftarrow ind_w$;
  end for
  repeat
    for $w \leftarrow 1$ to γ **do**
      $y \leftarrow$ número inteiro aleatório entre 1 e μ;
      $ind_w \leftarrow Pop_y$;
      $ind_w \leftarrow$ mutacaoParametros(ind_w, σ_{real}, $\sigma_{binomial}$);
      $Filho_w \leftarrow$ mutacaoIndividuo(ind_w);
    end for
    for $w \leftarrow 1$ to κ **do**
      $y \leftarrow$ número inteiro aleatório entre 1 e γ;
      $Filho_y \leftarrow$ III($Filho_y$); {Busca Local}
    end for
    $Pop \leftarrow$ selecao(Pop, Filhos, μ);
  until Critério de parada ser satisfeito
  for $w \leftarrow 1$ to μ **do**
    $ind_w \leftarrow$ LocalSearchFullRepresentation(ind_w);
  end for
  return melhorIndividuoPop();

D. Greedy-Random NEH

O procedimento Greedy-Random NEH é responsável por gerar uma parcela de indivíduos para a população inicial para o algoritmo proposto. Ele é uma adaptação do procedimento NEH e funciona como segue. Inicialmente todas as tarefas são inseridas na Lista de Candidatos (LC). Na primeira iteração são selecionadas, aleatoriamente, duas tarefas $j \in LC$ e executado o procedimento que procura o melhor sequenciamento possível entre essas duas tarefas. Nesse sequenciamento a tarefa escolhida é executada na máquina que possa terminá-la o mais cedo. Na segunda iteração essas duas tarefas são retiradas da LC. Uma única tarefa da LC é, então, escolhida aleatoriamente, sendo novamente aplicado o procedimento NEH Adaptado, desta vez para verificar a melhor posição de inserção da tarefa escolhida. O processo continua com a retirada da tarefa escolhida da LC e a aplicação do procedimento NEH até que todas as tarefas tenham sido sequenciadas.

E. Iterated Greedy

O procedimento Iterated Greedy (IG), cujo pseudocódigo está esquematizado no Algoritmo 2, é utilizado para gerar uma parcela de indivíduos para compor a população inicial do algoritmo proposto.

Na linha 1 do Algoritmo 2 é gerada uma solução inicial com o método NEH de [33]. Este método funciona como segue. Inicialmente, calcula-se o tempo médio de processamento em cada estágio para cada tarefa, isto é, se uma tarefa j pode passar por duas máquinas no estágio i e, essas máquinas consomem os tempos p_{ij1} e p_{ij2}, então a essa tarefa j será atribuído o tempo médio, calculado como $\bar{p}_{ij} = \frac{p_{ij1}+p_{ij2}}{2}$. A seguir, é calculado, para cada tarefa j, o somatório desses tempos médios de processamento em todos os estágios, isto é, é calculado o tempo médio de processamento da tarefa j, $\bar{p}_j = \sum_i \bar{p}_{ij}$. Na primeira iteração são selecionadas as duas tarefas com maiores $\bar{p}_j$ e realizado um procedimento que procura o melhor sequenciamento possível entre essas duas tarefas. Neste sequenciamento a tarefa escolhida é executada na máquina que possa terminá-la o mais cedo. Este sequenciamento é usado como base para inserir a terceira tarefa. Na segunda iteração é, então, escolhida a terceira tarefa com o maior valor de $\bar{p}_j$ e sequenciada na melhor posição possível. O processo continua até que todas as tarefas tenham sido sequenciadas.

Na linha 2 do Algoritmo 2 é aplicado o procedimento de busca local III de [34].

Da linhas 3 à 16 do Algoritmo 2, as fases de destruição, reconstrução e aceitação da solução são repetidas até que o critério de parada seja atendido. A destruição (linhas 6 à 8) consiste em retirar d (parâmetro de entrada) tarefas da solução corrente e inseri-las em um vetor π_R. Para reconstruir a solução (linhas 9 à 11), cada tarefa de π_R é inserida na melhor posição possível do sequenciamento. Depois é aplicado novamente o método de busca local III. Após isso, se a solução gerada for melhor que a solução corrente, então a solução corrente é atualizada.

Algorithm 2 IG

Require: d, $Temperatura$, $criterioParada$
Ensure: π
  $\pi \leftarrow$ construcaoNEH();
  $\pi \leftarrow$ III(π); {Busca Local}
  repeat
    $\pi' \leftarrow \pi$;
    $\pi_R \leftarrow \emptyset$;
    for $w \leftarrow 1$ to d **do**
      Retire uma tarefa aleatoriamente de π' e a insira em π_R;
    end for
    for $w \leftarrow 1$ to d **do**
      Retire a tarefa $\pi_R(w)$ e a insira na melhor posição possivel de π';
    end for
    $\pi' \leftarrow$ III(π'); {Busca Local}
    if $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
      $\pi \leftarrow \pi'$;
    end if
  until *criterioParada* ser satisfeito;
  return π

F. Busca Local com Representação Completa

Ao observar o sequenciamento, é notavel que o makespan é determinado por um caminho crítico. Por exemplo, na Figura 1, nota-se que o inicio da última operação da tarefa 5 é determinado pelo fim da operação da mesma tarefa no estágio anterior, e o inicio dessa operação é determinado pelo fim da primeira operação da tarefa 1. Formando um caminho de operações. Uma operação é a execução de uma tarefa em um estágio. Quando esse caminho determina o makespan, ele é chamado de caminho crítico.

Segundo [35], melhorar o valor makespan movendo uma operação que não está no caminho crítico é muito improvável. Portanto, só se aplica a busca local para as operações no caminho crítico.

Esse método de busca local se inicia com a operação cujo o momento de conclusão é igual ao valor do makespan, na solução corrente. No exemplo da Figura 1 essa é a tarefa 5 na máquina 4. Essa operação é realocada em todas as posições possiveis de sequenciamento em todas as máquinas elegíveis. Se houver melhora na solução, a solução corrente é atualizada, e o procedimento reinicializado. Não havendo melhora a próxima operação no caminho crítico é considerada. Essa pode ser a operação do estágio anterior da mesma tarefa ou a operação anterior, de uma outra tarefa, na mesma máquina. Quando duas ou mais tarefas determinar o maksepan ou o tempo de início de outra tarefa no caminho crítico, a busca do local é interrompida. Neste caso, o valor makespan não pode ser melhorado por meio da realocação de uma única operação. O procedimento se encerra quando não houver mais tarefas no caminho crítico.

V. Experimental Results

Os dois algoritmos propostos foram implementados em C++, utilizando-se a IDE Borland C++ Builder 6. Os testes foram executados em um computador Intel Core 2 Quad, 2.67GHz, com 4 GB de memória RAM, sob sistema operacional Windows 7 32 bits.

Para testá-los foram utilizadas 8 famílias de instâncias, de [1], cujas características principais estão mostradas na Tabela VI. Nesta Tabela, n é a quantidade de máquinas, m é o número de estágios e m_i o número de máquinas em cada estágio. Como em cada família há 12 instâncias, há um total de 96 instâncias.

A Tabela VII mostra os valores dos parâmetros adotados no algoritmo. Nessa tabela as colunas μ, γ e κ mostram os valores dos parâmetros para o algoritmo ES, as colunas d e T mostram os valores dos parâmetros para o algoritmo IG. O parâmetro d é um número inteiro e aleatório. Esses valores foram definidos de forma empírica.

Tabela VI
FAMÍLIA DE INSTÂNCIAS

Família de Instâncias	n	m	m_i
15_2_1	15	2	1
15_2_3	15	2	3
15_3_1	15	3	1
15_3_3	15	3	3
50_4_2	50	4	2
50_4_4	50	4	4
50_8_2	50	8	2
50_8_4	50	8	4

Tabela VII
PARÂMETROS DOS ALGORITMO

Algoritmo	μ	γ	κ	d	T
ES	30	60	10	-	-
IG	-	-	-	2-8	0.5

O algoritmo proposto foi comparado com o melhor resultado da literatura de [31] e [35] em relação a três aspectos: 1) Soluções geradas na construção, 2) Soluções geradas durante as gerações e 3) Soluções geradas após aplicação da busca local com representação completa. Para avaliar o primeiro aspecto, calculou-se o *gap* das melhores soluções e das soluções médias geradas pelos algoritmos de construção para cada grupo de instâncias em relação aos valores ótimos (no caso das instâncias envolvendo 15 tarefas) ou melhores valores da literatura (no caso das instâncias de 50 tarefas). Para avaliar o segundo aspecto calculou-se o *gap* das melhores soluções e das soluções médias nas gerações do algoritmo baseado em ES. Para o terceiro aspecto calculou-se o o *gap* das soluções geradas após aplicação da busca local com representação completa.

O algoritmo foi executado 10 vezes para cada problema teste. Toda vez que o algoritmo ficava 40 iterações seguidas sem melhorar a solução a execução era interrompida.

A Tabela VIII mostra os resultados da comparação do algoritmo proposto com relação aos três aspectos apontados para as instâncias de 15 tarefas. A Tabela IX mostra os resultados da comparação do algoritmo proposto para as instâncias de 50 tarefas. Na primeira coluna dessas tabelas são apresentados os conjuntos de instâncias; nas colunas "Melhor" são apresentados os valores que se referem ao *gap* dos melhores valores para o *makespan* da variante, nas colunas "Médio", o *gap* dos valores médios do *makespan*. Nas colunas "Tempo" são apresentados os valores médios dos tempos de execução (em milisegundos). Nessa tabela, também é apresentado o percentual de melhoria das soluções finais com as soluções construídas.

Para as instâncias de 15 tarefas, na média, o algoritmo proposto apresentou um *gap* em relação aos melhores valores de 0.62%, e em relação aos valores médios de 1.16%. Para as instâncias de 50 tarefas, o algoritmo proposto apresentou um *gap* em relação aos melhores valores de -5.18%, e em relação aos valores médios de -4.06%.

Para as instâncias de 15 tarefas, na média, a fase das gerações e da busca local com representação completa conseguiu melhorar em 2.94% em relação as melhores soluções geradas na fase de construção. Para as instâncias de 50 tarefas, a fase das gerações e da busca local com representação completa conseguiu melhorar em 4.21% em relação as melhores soluções geradas na fase de construção.

Dos 48 ótimos globais conhecidos (problemas-teste de 15 tarefas), o algoritmo proposto foi capaz de encontrar 33 deles. Nos 48 problemas-teste cujo as soluções ótimas não são conhecidas o algoritmo proposto conseguiu melhorar os resultados da literatura em 34 deles.

VI. CONCLUSIONS AND FUTURE WORK

Este trabalho tratou o problema de sequenciamento *flowshop* híbrido e flexível, tendo como objetivo minimizar o *makespan*. Em vista de sua complexidade, ele foi resolvido por meio de um algoritmo baseado em Estratégias Evolutivas.

No algoritmo proposto as soluções iniciais são geradas pela aplicação de uma adaptação da regra de avaliação NEH, e pela metaheurística IG. O principal mecanismo de busca no espaço de soluções do algoritmo desenvolvido é a mutação, que é feita com base em dois vetores, sendo um relativo à probabilidade de aplicar cada tipo de mutação, e outro relativo a intensidade em que cada mutação é aplicada. Em ambos os algoritmos, feita a mutação, uma parcela da população de filhos é refinada por um procedimento de busca local baseado no método *Iterative Improvement Insertion* (III). Por fim, os indivíduos da população final são refinados por um método de busca local com representação completa.

Comos trabalhos futuros propõe-se testar a geração de soluções iniciais com outras regras de avaliação, desenvolver outros tipos de mutação, bem como testar o desempenho do algoritmo em problemas-teste de maior porte.

VII. ACKNOWLEDGMENTS

Os autores agradecem ao CEFET-MG e às agências CAPES, CNPq e FAPEMIG pelo apoio ao desenvolvimento deste trabalho.

REFERÊNCIAS

[1] R. Ruiz, F. Sivrikaya, and T. Urlings, "Modeling realistic hybrid flexible flowshop scheduling problems," *Computers & Operations Research*, vol. 35, pp. 1151–1175, 2008.

[2] H. G. Beyer and H. P. Schwefel, "Evolution strategies - a comprehensive introduction," *Natural Computing*, vol. 1, pp. 3–52, 2002.

[3] L. Costa and P. Oliveira, *Manual de Computação Evolutiva e Metaheurísticas*, 1st ed. Imprensa da Universidade de Coimbra e Editora da UFMG, 2012, vol. 1, ch. Estratégias Evolutivas, pp. 49–65.

[4] V. Coelho, M. Souza, I. Coelho, F. Guimaraes, and B. Coelho, "Estratégias evolutivas aplicadas a um problema de programação inteira mista," in *Anais do X Congresso Brasileiro de Inteligência Computacional*, Fortaleza, novembro de 2011, 8p, 2011.

[5] L. Costa and P. Oliveira, "Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems," *Computers & Chemical Engineering*, vol. 25, no. 10, pp. 257–266, 2001.

[6] M. F. Tasgetiren, Q.-K. Pan, P. Suganthan, and O. Buyukdagli, "Variable iterated greedy algorithm with differential evolution for solving no-idle flowshops," in *SIDE-EC 2012*, Zakopane, Poland, 8p, 2012.

[7] L. Burtseva, R. Romero, S. Ramirez, V. Yaurima, F. González-Navarro, and P. Perez, *Lot processing in Hybrid Flow Shop scheduling problem*, ser. Production Scheduling. InTech, 2012, pp. 65–96.

Tabela VIII
Gap DO ALGORITMO PROPOSTO NAS INSTÂNCIAS DE 15 TAREFAS

	Construção			Gerações			Representação Completa			Melhoria	
Instância	Melhor	Médio	Tempo(ms)	Melhor	Médio	Tempo(ms)	Melhor	Médio	Tempo(ms)	Melhor	Médio
15_2_1	0.27	0.44	522.85	0.20	0.21	5894.00	0.14	0.18	5910.23	0.09	0.33
15_2_3	2.20	17.44	1475.50	1.12	2.30	6998.93	0.68	1.87	7890.03	4.25	11.70
15_3_1	0.36	0.60	762.38	0.30	0.31	5515.41	0.27	0.29	5555.73	0.11	0.31
15_3_3	10.99	14.46	2248.85	1.46	2.31	11343.88	1.41	2.31	11343.88	7.30	9.01
Média	3.45	8.23	-	0.77	1.28	-	0.62	1.16	-	2.94	5.34

Tabela IX
Gap DO ALGORITMO PROPOSTO NAS INSTÂNCIAS DE 50 TAREFAS

	Construção			Gerações			Representação Completa			Melhoria	
Instância	Melhor	Médio	Tempo(ms)	Melhor	Médio	Tempo(ms)	Melhor	Médio	Tempo(ms)	Melhor	Médio
50_4_2	5.51	6.71	17370.47	2.06	3.23	711717.63	1.86	3.16	719918.29	3.24	3.62
50_4_4	0.59	1.88	24335.90	**-3,44**	**-2.20**	687512.55	**-4.12**	**-2.63**	735382.93	4.65	4.65
50_8_2	1.76	2.23	36694.32	**-5.54**	**-4.65**	1404418.00	**-5.83**	**-4.83**	1421702.78	4.28	4.30
50_8_4	1.13	1.52	51117.93	**-13.81**	**-12.62**	1217983.55	**-13.76**	**-12,87**	1338912.62	4.66	4.80
Média	2.25	3.08	-	**-5.18**	**-4.06**	-	**-5.46**	**-4.29**	-	4.21	4.09

[8] I. Ribas, R. Leisten, and J. Framiñan, "Review and classification of hybrid flowshop scheduling problems from a production system and a solutions procedure perspective," *Computers & Operations Research*, vol. 37, pp. 1439–1454, 2010.

[9] M. Salvador, "A solution to a special class of flowshop scheduling problems," in *Symposium on the theory of scheduling and its applications*. Berlin:Springer, 1973, pp. 83–91.

[10] M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP-completness*. San Francisco: Freeman, 1979.

[11] V. Oduguwa, A. Tiwari, and R. Roy, "Evolutionary computing in manufacturing industry: an overview of recent applications," *Applied Soft Computing*, vol. 5, pp. 281–299, 2005.

[12] S. Parthasarathy and C. Rajendran, "An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs," *International Journal of Production Economics*, vol. 49, pp. 255–263, 1997.

[13] Y. Sun, C. Zhang, L. Gao, and X. Wang, "Multi-objective optimization algorithms for flow shop scheduling problem: a review and prospects," *International Journal of Advanced Manufacturing Technology*, vol. 55, pp. 723–739, 2010.

[14] V. Botta-Genoulaz, "Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness," *International Journal of Production Economics*, vol. 64, pp. 101–111, 2000.

[15] M. Kurz and R. Askin, "Comparing scheduling rules for flexible flow lines," *International Journal of Production Economics*, vol. 85, pp. 371–388, 2003.

[16] B. Naderi and R. Ruiz, "The distributed permutation flowshop scheduling problem," *Computers & Operations Research*, vol. 37, pp. 754–768, 2010.

[17] R. Ruiz and C. Andrés-Romano, "Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times," *International Journal of Advanced Manufacturing Technology*, vol. 57, pp. 777–794, 2011.

[18] R. Ruiz and C. Maroto, "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility," *European Journal of Operational Research*, vol. 169, pp. 781–800, 2006.

[19] R. Ruiz and J. Vázquez-Rodríguez, "The hybrid flow shop scheduling problem," *European Journal of Operational Research*, vol. 205, pp. 1–18, 2010.

[20] T. Nishi, Y. Hiranaka, and M. Inuiguchi, "Lagrangian relaxation with cut generation for hybrid flowshop scheduling problems to minimize the total weighted tardiness," *Computers & Operations Research*, vol. 37, pp. 189–198, 2010.

[21] A. Azzi, M. Faccio, A. Persona, and F. Sgarbossa, "Lot splitting scheduling procedure for makespan reduction and machine capacity increase in a hybrid flow shop with batch production," *International Journal of Advanced Manufacturing Technology*, vol. 59, pp. 775–786, 2011.

[22] M. Akhshabi, M. Akhshabi, and J. Khalatbari, "Proposing a clonal selection algorithm for hybrid flow shop," *African Journal of Business Management*, vol. 6, pp. 5744–5749, 2012.

[23] A. Ziaeifar, R. Tavakkoli-Moghaddam, and K. Pichka, "Solving a new mathematical model for a hybrid flow shop scheduling problem with a processor assignment by a genetic algorithm," *International Journal of Advanced Manufacturing Technology*, vol. 56, pp. 1–11, 2011.

[24] V. Yaurima, L. Burtseva, and A. Tchernykh, "Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers," *Computers & Industrial Engineering*, vol. 56, pp. 1452–1463, 2009.

[25] M. Boudhar and N. Meziani, "Two-stage hybrid flow shop with recirculation," *International Transactions in Operational Research*, vol. 17, pp. 239–255, 2010.

[26] S. Carpov, J. Carlier, D. Nace, and R. Sirdey, "Two-stage hybrid flowshop with precedence constraints and parallel machines at second stage," *Computers & Operations Research*, vol. 39, pp. 736–745, 2012.

[27] B. Tadayon and N. Salmasi, "A flexible flowshop scheduling problem with machine eligibility constraint and two criteria objective function," *Engineering and Technology*, vol. 62, pp. 103–108, 2012.

[28] ——, "A two-criteria objective function flexible flowshop scheduling problem with machine eligibility constraint," *International Journal of Advanced Manufacturing Technology*, vol. 60, pp. 1–15, 2012.

[29] B. Naderi, R. Ruiz, and M. Zandieh, "Algorithms for a realistic variant of flowshop scheduling," *Computers & OperationsResearch*, vol. 37, pp. 236–246, 2010.

[30] F. M. Defersha and M. Chen, "Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem," *Internationa Journal of Advanced Manufacturing Technology*, vol. 57, pp. 1–17, 2011.

[31] T. Urlings and R. Ruiz, "Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems," *International Journal of Metaheuristics*, vol. 1, pp. 30–54, 2010.

[32] M. Zandieh, E. Mozaffari, and M. Gholami, "A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems," *Journal of Intelligent Manufacturing*, vol. 21, pp. 731–743, 2010.

[33] M. Nawaz, E. E. Jr, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *The International Journal of Management Science*, vol. 11, pp. 91–95, 1983.

[34] R. Ruiz and T. Stützle, "An iterated greedy algorithm for the flowshop problem with sequence dependent setup times," in *The 6th Metaheuristics International Conference*, 2005, pp. 817–826.

[35] T. Urlings, R. Ruiz, and T. Stützle, "Shifting representation search for hybrid flexible flowline problems," *European Journal of Operational Research*, vol. 207, pp. 1086–1095, 2010.