UM ALGORITMO HEURÍSTICO MULTIOBJETIVO BASEADO NO MÉTODO DE NEWTON PARA O PROBLEMA INTEGRADO DE ALOCAÇÃO DE RECURSOS E SEQUENCIAMENTO DE OPERAÇÕES

Miguel Angel Fernández Pérez
fernandezmiguelp@gmail.com

Fabricio Oliveira
fabricio.oliveira@puc-rio.br

Fernanda Maria Pereira Raupp
fraupp@puc-rio.br

Departamento de Engenharia Industrial
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, Gávea - Rio de Janeiro, RJ- Brasil - 22451-900

RESUMO

Neste artigo é apresentado um algoritmo heurístico multiobjetivo para o problema integrado de alocação de recursos e seqüenciamento de operações (iRS/OS, do inglês *integrated Resource Selection and Operation Sequences*), com o objetivo de minimizar o makespan e o balanceamento de carga de trabalho. O algoritmo proposto é uma adaptação do método de Newton para problemas de otimização multiobjetivo com variáveis continuas, pertencente à classe de métodos de descida. Nesta adaptação, o algoritmo visa em primeiro lugar, melhorar a alocação de recursos (máquinas) e depois a seqüência de operações de forma recursiva, repetindo este procedimento para várias soluções iniciais para encontrar no final, um conjunto de soluções não-dominadas em relação às medidas de desempenho. Experimentos numéricos são apresentados para mostrar o potencial do algoritmo quando comparado com resultados conhecidos de algoritmos existentes para as mesmas instâncias teste.

PALAVRAS CHAVE. Problema integrado de alocação de recursos e seqüenciamento de operações, Otimização multiobjetivo, Método de Newton.

Área principal: OC - Otimização Combinatória

ABSTRACT

In this paper we propose a hierarchical heuristic algorithm for the integrated Resource Selection and Operation Sequences (iRS/OS) with the objective of minimizing the makespan and balancing workload. The proposed method is an adaptation of Newton's method for multiobjective optimization problems with continuous variables, belonging to the class of descent methods. In this adaptation, the method aims primarily at improving the allocation of machines, then the sequence of operations and is recursively repeated for several initial solutions to find a set of non-dominated solutions with respect to performance measures. Numerical results are presented to show the potential of the algorithm when compared with the known results of available algorithms for the same test instances.

KEYWORDS. Integrated resource selection and operation sequences problem, Multiobjective optimization, Newton's method.

Main area: OC - Combinatorial Optimization

1. Introdução

O problema integrado de alocação de recursos e seqüenciamento de operações (iRS/OS, do inglês *integrated Resource Selection and Operation Sequences*), é derivado de um processo de produção real num sistema de manufatura, onde, cada pedido está associado a um conjunto de operações, as quais não têm seqüência fixa e depende das restrições de precedência. O escalonamento do processo de produção deve considerar os tamanhos de lote dos diferentes pedidos, tempos de transporte, disponibilidade dos recursos e outras características do sistema de manufatura (ZHANG *ET AL*., 2006).

O problema iRS/OS é uma extensão dos problemas de escalonamento *flow shop*, *job shop* e *job shop* flexível, o qual consiste em determinar a seqüência factível de processamento de um conjunto de pedidos de clientes com seus respectivos tamanhos de lote a serem realizados sobre um conjunto de máquinas, considerando a existência de um conjunto de máquinas disponíveis a serem alocadas para cada operação, visando otimizar uma ou mais medidas de desempenho, geralmente associadas ao fator tempo e ao balanceamento de uso das máquinas, onde as operações de cada pedido seguem uma restrição precedência.

Tan e Khoshnevis (2000, 2004) abordaram o tema da integração do planejamento e escalonamento de processos, onde discutem as vantagens e dificuldades da integração, a aplicabilidade de vários enfoques e os algoritmos utilizados na literatura. Kolisch e Hess (2000) estudaram o problema de escalonamento em grande escala para o processo de montagem, considerando recursos e restrições na disponibilidade de peças, onde os diferentes pedidos dos clientes precisam dos mesmos tipos de peças e a fabricação de peças deve levar em conta as decisões do tamanho de lote. Tais autores desenvolveram três métodos heurísticos para resolver o problema; um método aleatório amostral e dois métodos baseados em busca tabu.

Vários trabalhos têm sido realizados no sentido de resolver o problema iRS/OS. Moon *et al*. (2002) propõem um algoritmo genético como o objetivo de minimizar a soma da carga de trabalho das máquinas e o tempo de transporte total, enquanto a carga de trabalho das máquinas é balanceada. No entanto, a codificação do cromossomo só considera a informação da seqüência de operações, enquanto para a alocação de recursos é utilizada uma heurística baseada no tempo de processamento mínimo, a qual, no entanto, pode não atingir uma solução ótima. Um trabalho similar, mas considerando o *outsourcing* das operações foi desenvolvido por Lee *et al*. (2002), no qual foi aplicado um algoritmo genético, com o objetivo é minimizar o makespan cumprindo o prazo de entrega de cada pedido. Yan *et al*. (2007) propuseram um algoritmo genético com um novo operador de mutação, baseado em perturbação e busca local, com a finalidade de minimizar o makespan, enquanto asseguram a restrição do prazo de entrega dos pedidos com o auxílio de uma função aptidão que considera penalidade por atraso.

No que tange a trabalhos que levam em conta mais de um objetivo no problema iRS/OS; Zhang *et al*. (2006) propuseram o algoritmo genético baseado em múltiplos estágios das operações (moGA, do inglês *multistage operation-based Genetic Algorithm*). Este enfoque define o cromossomo como dois vetores que contém tanto a informação da seqüência de operações quanto à alocação de máquinas, considerando a minimização do makespan e o balanceamento da carga de trabalho. Yang e Tang (2009) desenvolveram um algoritmo genético multiobjetivo com uma estratégia adaptativa baseada nos ajustes da taxa de crossover e da taxa de mutação. A minimização simultânea do tempo inativo das máquinas e da penalidade por atraso/antecipação é considerada. Dayou *et al.* (2009) apresentaram um algoritmo genético multiobjetivo híbrido. Três objetivos foram simultaneamente minimizados: o makespan, o balanceamento da carga de trabalho e o tempo total de transporte entre máquinas. O algoritmo utiliza a estratégia de classificar as soluções não-dominadas mediante a aglomeração destas, junto com a busca local das soluções.

Este artigo tem como foco o desenvolvimento de um novo método computacional para resolver está classe de problemas de escalonamento de produção, que obtenha boas soluções em um tempo aceitável. O algoritmo proposto é uma adaptação do método de Newton para problemas de otimização multiobjetivo com variáveis continuas, pertencente à classe de métodos de descida, ou seja, que em cada passo melhora todas as funções objetivo na direção do

gradiente. Neste novo enfoque, o problema iRS/OS é considerado, tendo como objetivo minimizar simultaneamente o makespan e o balanceamento de carga de trabalho.

O artigo está divido como segue: na seção 2 são apresentados alguns conceitos sobre otimização multiobjetivo, a descrição do problema iRS/OS, a representação de soluções e a geração da vizinhança das soluções. Na seção 3 é descrito o algoritmo proposto para resolver o problema multiobjetivo iRS/OS a partir da adaptação do método de Newton. A seção 4 apresenta os resultados computacionais, os quais são comparados com resultados de métodos existentes. Finalmente, na seção 5, as conclusões e as considerações do algoritmo proposto são abordadas.

2. Descrição do problema multiobjetivo iR/OS

O problema de otimização multiobjetivo com r objetivos ($r >1$), pode ser definido da seguinte forma: dado o espaço de busca X em $\mathbb{R}^n$ de soluções factíveis, onde, deseja-se encontrar um vetor $x^* \in X$ que minimize simultaneamente as r funções objetivo $f(x) = \{f_1(x), f_2(x), \dots, f_r(x)\}$, onde não exista um $y \in X$, tal que:

$$f_i(y) \leq f_i(x^*) \text{ para todo } i = 1, \dots, r \text{ e } f_j(y) < f_j(x^*) \text{ para algum } j = 1, \dots, r$$

Onde x^* é chamada de solução não-dominada, solução Pareto-ótima ou solução eficiente. Desta forma, uma solução não-dominada não pode ser melhorada em relação a qualquer função objetivo sem piorar pelo menos outra função objetivo. O conjunto das soluções não-dominadas em X é chamado de conjunto Pareto-ótimo, e a imagem de um determinado conjunto Pareto-ótimo, no espaço dos valores dos objetivos, é chamada de Fronteira de Pareto (KONAK, COIT & SMITH, 2006).

O problema iRS/OS pode ser definido como segue: dado um conjunto de K pedidos, de tamanho de lote q_k $(k = 1, \dots, K)$ e um conjunto de R máquinas, onde cada pedido k e definido por uma série de J_k operações. Considere que cada operação o_{ki} $(k = 1, \dots, K,\ i = 1, \dots, J_k)$ deve ser processado por alguma máquina M_r $(r = 1, \dots, R)$ dentro do conjunto de máquinas M_{ki} capaz de processar o_{ki}. O tamanho de lote q_k é dividido em sub-lotes iguais, chamado de unidade de carga u_k para ser processado nas máquinas. Deseja-se encontrar a seqüência de operações dos pedidos e a alocação das operações nas máquinas, de tal forma de satisfazer as restrições de precedência entre as operações, à disponibilidade das máquinas e, além disso, seja ótima em relação à minimização do makespan e do balanceamento de carga de trabalho (ZHANG *ET AL.*, 2006). O makespan é dado como o tempo de conclusão de todos os pedidos nas máquinas e o balanceamento de carga de trabalho pela variância da carga de trabalho das máquinas.

As hipóteses consideradas no problema iRS/OS são:

- Todos os pedidos chegam simultaneamente.
- Todas as máquinas estão disponíveis no início.
- A unidade de carga é fixa para todas as máquinas, e as máquinas não podem parar até terminar de processar o tamanho de lote.
- O tempo de transporte entre máquinas é considerado.
- Para o mesmo pedido, o processamento numa operação começa logo que a unidade de carga da operação predecessora tenha sido concluída e transportada.
- O tempo de setup está incluído no tempo de processamento.

A seguir é apresentado um exemplo para o problema iRS/OS proposto por Zhang *et al.* (2006), o qual consiste em 2 pedidos, 7 operações e 5 máquinas. A Figura 1 mostra dois tipos de materiais a serem processados. Observa-se, que o Pedido 1 requer 3 operações, que o Pedido 2 requer de 4 operações e que ambos seguem restrições de precedência. O Pedido 1, tem tamanho de lote de 60 unidades do Produto 1, e o Pedido 2, tem tamanho de lote de 50 unidades do Produto 2. Para obter os produtos finais é preciso remover 7 componentes ou volumes dos dois materiais. A unidade de carga é 10 unidades, isto é, para o Pedido 1, o número de sub-lotes é de $60/10 = 6$ e para o Pedido 2 é de $50/10 = 5$ (em cada máquina o tamanho de lote será

processado em 10 unidades de cada vez). O plano de manufatura é mostrado na Tabela 1, a qual indica o tipo de cada operação e as máquinas disponíveis por operação.

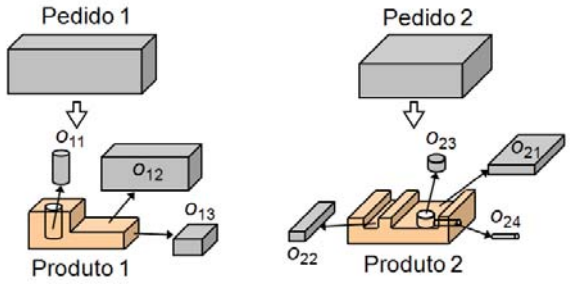

Figura 1 – Exemplo do problema iRS/OS

Pedido	Tamanho de lote	Índice da operação	Operação	Tipo de operação	Máquinas Disponíveis
1	60	1	o_{11}	Perfuração	M_1, M_4
		2	o_{12}	Fresagem	M_2, M_5
		3	o_{13}	Fresagem	M_2, M_3, M_5
2	50	4	o_{21}	Fresagem	M_2, M_3
		5	o_{22}	Fresagem	M_3, M_5
		6	o_{23}	Perfuração	M_1, M_4, M_5
		7	o_{24}	Perfuração	M_1, M_2

Tabela 1 – Plano de manufatura do exemplo iRS/OS

A Figura 2 mostra as restrições de precedência das operações. As seqüências de operações são muitas e devem respeitar as restrições de precedência. Por exemplo, para o Pedido 1, tanto $\{o_{11}, o_{12}, o_{13}\}$ quanto $\{o_{12}, o_{11}, o_{13}\}$ são seqüências factíveis, porém $\{o_{11}, o_{13}, o_{12}\}$ é infactível.

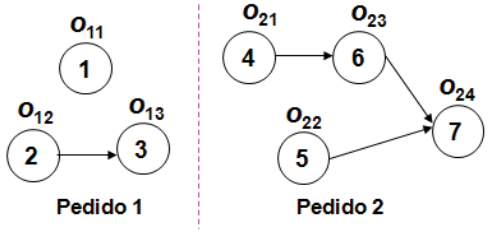

Figura 2 – Restrições de precedência do exemplo iRS/OS

A Tabela 2 mostra o tempo de processamento dado para cada operação. A Tabela 3 mostra o tempo de transporte dado entre diferentes máquinas.

	Pedido 1			Pedido 2			
Operação	1	2	3	4	5	6	7
Máquina	o_{11}	o_{12}	o_{13}	o_{21}	o_{22}	o_{23}	o_{24}
M_1	5	-	-	-	-	6	8
M_2	-	7	6	6	-	-	12
M_3	-	-	5	7	9	-	-
M_4	7	-	-	-	-	5	-
M_5	-	6	8	-	5	6	-

Tabela 2 – Tempo de processamento das operações em máquinas diferentes do exemplo iRS/OS

	M_1	M_2	M_3	M_4	M_5
M_1	-	7	27	26	17
M_2	19	-	15	19	10
M_3	5	17	-	36	27
M_4	8	20	4	-	30
M_5	18	18	14	5	-

Tabela 3 - Tempo de transporte entre máquinas diferentes do exemplo iRS/OS

Uma solução do problema iRS/OS é representada por dois vetores: o vetor de seqüência de operações s^* e o vetor de alocação de máquinas v^*. Para gerar a seqüência de operações, é considerado um nível de prioridade para o conjunto de operações, denotado por π. Seja J o número total de operações, então π é dado por uma permutação aleatória de $\{1, 2, \ldots, J\}$, ou seja, uma operação com prioridade 1 terá maior preferência no escalonamento que as operações com prioridade 2, 3, ..., J; uma operação com prioridade 2 terá maior preferência que as operações com prioridade 3, 4, ..., J, e assim por diante.

Agora, considerando o exemplo anterior com $J = 7$, pode-se ter, por exemplo, $\pi = \{5, 1, 7, 2, 4, 6, 3\}$ e utilizando o procedimento de seqüenciamento de operações, proposto por Dayou *et al*. (2009), que consiste em verificar as restrições de precedência entre as operações e dependo do nível de prioridade π, as operações são escalonadas. Para o exemplo, a seqüência de operações factível é dada por $s^* = \{o_{12}, o_{21}, o_{22}, o_{11}, o_{23}, o_{13}, o_{24}\}$, a qual pode ser representada através de seus índices como $s^* = \{2, 4, 5, 1, 6, 3, 7\}$, conforme mostra a Tabela 4. Agora que s^* é conhecido, pode-se gerar a alocação de máquinas, utilizando o procedimento proposto por Zhang *et al*. (2006), no qual, uma máquina disponível é alocada ao acaso para cada operação. Assim, a alocação de máquinas é dada por $v^* = \{5, 2, 3, 1, 4, 3, 2\}$, como se mostra na Tabela 4.

Operação	1	2	3	4	5	6	7
π	5	1	7	2	4	6	3
s^*	2	4	5	1	6	3	7
v^*	5	2	3	1	4	3	2

Tabela 4 - Exemplo de uma solução (s^*, v^*)

Segundo Xia e Wu (2005), em um algoritmo de busca local, o tipo de vizinhança pode influenciar fortemente o desempenho do algoritmo. Embora a escolha de uma vizinhança que contém um grande número de soluções candidatas aumente a probabilidade de encontrar boas soluções, o tempo computacional para encontrar vizinhos disponíveis também irá aumentar.

Para o problema em questão, a vizinhança para alocação de máquinas pode ser gerada através da atribuição do conjunto de máquinas $r \in M_{ki}$ restantes para cada operação o_{ki}. Se J é o número total de operações e R é o número de máquinas, então, o número máximo de soluções vizinhas é dado por $J \times (R - 1)$, já que o problema pode ser parcial ou totalmente flexível. Para a

geração de soluções vizinhas da seqüência de operações, uma técnica simples consiste na troca entre duas operações de uma seqüência, que gerem sequências factíveis de operações, isto é, que respeitem as restrições de precedência. Note que, se J é o número total de operações, então, o número máximo de soluções de vizinhas é dado por $(J) \times (J-1)/2$. Para nosso exemplo com 7 operações ($J = 7$) o número total de permutações geradas a partir de s^* é $(7) \times (6)/2 = 21$, porém somente 12 seqüências são factíveis.

3. Método de Newton multiobjetivo

Desenvolvido por Fliege *et al.* (2009), este método é utilizado para resolver problemas multiobjetivos com variáveis contínuas sem restrições. Sua característica principal é pertencer à classe dos algoritmos de descida *multi-start*, isto é, que são gerados vários pontos iniciais que são melhorados recursivamente em relação às funções objetivo.

3.1 Direção de Newton

Dada uma função $F: U \subset R^n \rightarrow R^m$ duas vezes continuamente diferenciável e um ponto não estacionário $x \in U$, a direção de Newton em x, denotada por $s(x)$, é obtida resolvendo-se o seguinte problema:

$$\begin{cases} \min \max_{j=1,\dots,m} \nabla F_j(x)^T s + \frac{1}{2} s^T \nabla^2 F_j(x) s \\ s.t. \;\; s \in \mathbb{R}^n. \end{cases}$$

Denote por $\theta(x)$ o valor ótimo do problema, que é dado por:

$$\theta(x) = \inf_{s \in \mathbb{R}^n} \max_{j=1,\dots,m} \nabla F_j(x)^T s + \frac{1}{2} s^T \nabla^2 F_j(x) s$$

a direção de Newton é dada por:

$$s(x) = \operatorname*{argmin}_{s \in \mathbb{R}^n} \max_{j=1,\dots,m} \nabla F_j(x)^T s + \frac{1}{2} s^T \nabla^2 F_j(x) s$$

Assim, resolvendo este problema recursivamente e obtendo $s(x_k)$ e $\theta(x_k)$ em cada passo k, pode-se fazer $x_{k+1} = x_k + s(x_k)$, até que $\theta(x_k) = 0$ com certa tolerância, ou seja, quando não é seja possível continuar melhorando as funções objetivo.

A Figura 3 mostra a intepretação gráfica do Método de Newton para duas funções. Note que $\theta(x)$ sempre é negativo, garantindo sempre a minimização de todos os objetivos.

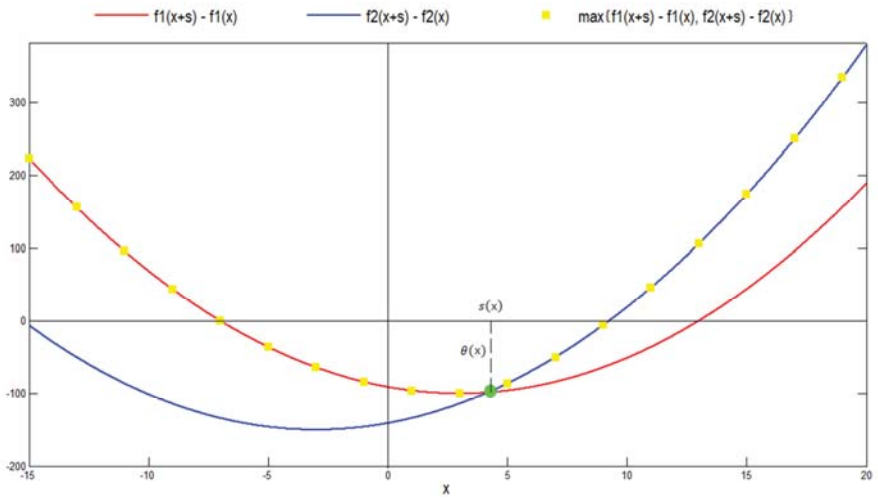

Figura 3 – Intepretação gráfica do Método de Newton

4. Algoritmo heurístico para o problema multiobjetivo iRS/OS

A seguir será apresentado o algoritmo heurístico hierárquico para o problema multiobjetivo iRS/OS baseado no Método de Newton.

Neste artigo, uma adaptação deste método foi realizada para considerar um domínio discreto e ainda considerar um novo critério de parada, pelo fato que o método original utiliza a condição necessária de primeira ordem para otimização multiobjetivo, que agora não pode ser mais usado já que a noção de gradiente de funções não se aplica no domínio discreto.

O algoritmo proposto não requer parâmetros para ponderar os objetivos e visa melhorar a alocação de máquinas em primeiro lugar e depois a seqüência de operações de forma recursiva. Este procedimento é repetido para várias soluções iniciais, obtendo no final um conjunto de soluções não-dominadas em relação às medidas de desempenho.

Uma aplicação deste algoritmo heurístico foi apresentado por Fernández e Raupp (2014), tratando o problema *job shop* flexível, considerando ainda a otimização de três objetivos, onde foram realizados vários experimentos numéricos e a comparação dos resultados obtidos com resultados conhecidos de algoritmos existentes na literatura, demostrando o potencial do algoritmo.

4.1 Estrutura principal

O algoritmo proposto parte uma solução inicial, ou seja, uma seqüência de operações e uma alocação de máquinas (s^*, v^*), que seja factível em relação à precedência das operações. O procedimento necessário para gerar a solução inicial foi descrito na seção 2, o algoritmo segue buscando novas soluções enquanto for possível melhorar pelo menos um dos objetivos. A estrutura principal do método proposto é apresentada no Pseudocódigo 1.

PSEUDOCÓDIGO 1: Main structure (NS)

```
ND ← ∅; S ← ∅;
for i = 1, ..., NS do
    Generate a initial solution (s*, v*);
    improvement ← TRUE;
    while improvement = TRUE do
        v' ← v*;
        Improve the machine assignment (s*, v*);
        if v' = v* then
            improvement ← FALSE;
        else
            Improve the operations sequence (s*, v*);
        end
    end
    S ← S ∪ {(s*, v*)};
end
ND ← non dominated solutions in S;
return (ND);
```

No pseudocódigo, NS denota o número de soluções geradas inicialmente e que serão melhoradas; S denota o conjunto de soluções melhoradas de todas as soluções inicialmente geradas; ND denota conjunto de soluções não-dominadas do conjunto S. Se v^* não muda $(v' = v^*)$, significa que nenhum dos objetivos pode ser melhorado. A seguir, são detalhados os componentes ou subrotinas da estrutura principal *(Main Structure)*.

4.2 Procedimento Melhorar Alocação de Máquinas

Considerando s^* fixa, a partir de uma alocação inicial de máquinas $v_0 = v^*$, para $k = 0$, este procedimento trata de melhorar em cada iteração pelo menos uma função objetivo sem piorar a outra função.

Para isso, em cada iteração k, denotamos por $V(v_k)$ a vizinhança da alocação v_k, e por v um vizinho de v_k ($v \in V(v_k)$). Além disso, denotamos por θ como o valor mínimo das variações máximas das funções objetivo f_1 e f_2 da solução corrente (s^*, v_k) e as soluções vizinhas (s^*, v), i.e.,

$$\theta = \min_{v \in V(v_k)} \max_{j=1,2} (f_j(s^*, v) - f_j(s^*, v_k))$$

Desta forma, como no método de Fliege *et al*. (2009), se θ for negativo, significa que é possível reduzir simultaneamente f_1 e f_2. Se θ for positivo, então, pelo menos uma função objetivo piora. No entanto, no domínio discreto, se θ for zero, então é possível melhorar no máximo um dos objetivos sem piorar o outro ou nenhum objetivo melhora.

Denotamos por df_1 e df_2 os conjuntos das variações máximas não-positivas de f_1 e f_2 respectivamente. Após a determinação de df_1 e df_2, o critério de parada é verificado, ou seja, se $\min_{v \in V(v_k)} \min_{j=1,2} df_j(v)$ é zero, então não é possível continuar melhorando nenhum objetivo, fazendo com que o procedimento termine. No caso em que o valor mínimo seja diferente de zero, então é possível melhorar um objetivo sem piorar o outro.

Em termos gerais, em cada iteração k, uma vizinhança de v_k é gerada e dependendo do valor de θ, o melhor vizinho de $V(v_k)$ é escolhido, de modo que pelo menos um objetivo é minimizado. O procedimento continua enquanto for possível reduzir pelo menos uma função objetivo (caso em que descent $= TRUE$). Nesse caso, k é atualizado com $k + 1$. Ao término do procedimento, a melhor alocação encontrada v_k será atribuída a v^*. A vizinhança $V(v_k)$ é gerada pelo método descrito na seção 2. O esquema geral deste procedimento, considerando duas medidas de desempenho, é apresentado no Pseudocódigo 2.

PSEUDOCÓDIGO 2: Improve the machines allocation (s^*, v)

$v_0 \leftarrow v^*;\ k \leftarrow 0;$ descent $\leftarrow TRUE;$
while descent $= TRUE$ **do**
  $df_1 \leftarrow \emptyset;\ df_2 \leftarrow \emptyset;$
  $\theta \leftarrow \min_{v \in V(v_k)} \max_{j=1,2}(f_j(s^*, v) - f_j(s^*, v_k));$
  if $\theta < 0$ **then**
    $v_{k+1} \leftarrow \underset{v \in V(v_k)}{argmin} \max_{j=1,2}(f_j(s^*, v) - f_j(s^*, v_k));$
  else if $\theta = 0$ **then**
    foreach $v \in V_{(v_k)}$ **do**
      if $\max_{j=1,2}(f_j(s^*, v) - f_j(s^*, v_k)) = 0$ **then**
        $df_j \leftarrow df_j \cup \{f_j(s^*, v) - f_j(s^*, v_k)\}, j = 1,2;$
      else
        $df_j \leftarrow df_j \cup \{0\}, j = 1,2;$
      end
    end
    if $\min_{v \in V(v_k)} \min_{j=1,2}(df_j(v)) = 0$ **then** descent $\leftarrow FALSE;$
    else if $\min_{v \in V(v_k)} df_1(v) = 0$ **then** $v_{k+1} \leftarrow \underset{v \in V(v_k)}{argmin}\, df_2(v);$
    else if $\min_{v \in V(v_k)} df_2(v) = 0$ **then** $v_{k+1} \leftarrow \underset{v \in V(v_k)}{argmin}\, df_1(v);$
    else $v_{k+1} \leftarrow \underset{v \in V(v_k)}{argmin}\, df_1(v);$
    end
  else descent $\leftarrow FALSE;$
  end
  if descent $= TRUE$ **then** $k \leftarrow k + 1;$ **end**
end
$v^* \leftarrow v_k;$
return (v^*);

4.3 Procedimento Melhorar Seqüência de Operações

Este procedimento resolve um problema auxiliar bi-objetivo, onde o primeiro objetivo é f_1 e o segundo objetivo é f_3, onde f_3 é um objetivo não conflitante com f_1. A ideia do procedimento é minimizar primeiramente f_1 e depois f_3 sem piorar f_1, com a finalidade de dar a oportunidade de se afastar de um possível mínimo local e encontrar uma melhor solução.

Inicialmente, considera-se v^* fixa, a partir de uma seqüência inicial de operações $s_0 = s^*$, para $k = 0$. Em cada iteração k, denotamos por $N(s_k)$ a vizinhança da seqüência s_k, e por s um vizinho de s_k ($s \in N(s_k)$). Neste caso, θ mede a variação de f_1 entre a solução corrente (s_k, v^*) e as soluções vizinhas (s, v^*). Se θ for negativo, então, é possível reduzir f_1. Se θ for positivo, então f_1 piora. Se θ for zero, então, é possível melhorar f_3 sem piorar f_1 ou nem f_1 ou f_3 melhoram.

Denotamos por df_3 o conjunto das variações máximas não-positivas de f_3. Após a determinação de df_3, verifica-se se $\min_{s \in N(s_k)} df_3(s)$ é zero, ou seja, não é possível continuar melhorando f_3. No caso em que o valor mínimo é diferente de zero, tem-se a situação em que é possível melhorar f_3 sem piorar f_1.

Em termos gerais, em cada iteração k, uma vizinhança de s_k é gerada e dependendo do valor de θ, escolhe-se o melhor vizinho de $N(s_k)$ de modo que f_1 seja minimizada (neste caso f_3 pode piorar). No caso que não seja possível melhorar f_1, escolhe-se o melhor vizinho de $N(s_k)$ que minimize f_3 sem piorar f_1. O procedimento continua enquanto for possível reduzir f_1 ou reduzir f_3 sem piorar f_1 (caso em que descent $= TRUE$). Nesse caso, k é atualizado com $k + 1$.

Finalmente, ao término do procedimento a melhor seqüência s_k será atribuída a s^*. A vizinhança $N(s_k)$ é gerada pelo método descrito na seção 2. O esquema geral deste procedimento é mostrado no Pseudocódigo 3.

PSEUDOCÓDIGO 3: Improve the operations sequence (s^*, v)

$s_0 \leftarrow s^*$; $k \leftarrow 0$; descent $\leftarrow TRUE$;
while descent $= TRUE$ **do**
  $df_3 \leftarrow \emptyset$;
  $\theta \leftarrow \min_{s \in N(s_k)} max(f_1(s, v^*) - f_j(s_k, v^*))$;
  if $\theta < 0$ **then**
    $s_{k+1} \leftarrow \underset{s \in N(s_k)}{argmin}\, max(f_1(s, v^*) - f_j(s_k, v^*))$;
  else if $\theta = 0$ **then**
    foreach $s \in N_{(s_k)}$ **do**
      if $\max_{j=1,3}(f_j(s, v^*) - f_j(s_k, v^*)) = 0$ **then**
        $df_3 \leftarrow df_3 \cup \{f_3(s, v^*) - f_3(s_k, v^*)\}$;
      else
        $df_3 \leftarrow df_3 \cup \{0\}$;
      end
    end
    if $\min_{s \in N(s_k)} df_3(v) = 0$ **then** descent $\leftarrow FALSE$;
    else $s_{k+1} \leftarrow \underset{s \in N(s_k)}{argmin}\, df_3(s)$;
    end
  else descent $\leftarrow FALSE$;
  end
  if descent $= TRUE$ **then** $k \leftarrow k + 1$; **end**
end
$s^* \leftarrow s_k$;
return (s^*);

5. Experimentos Numéricos

A implementação foi realizada em MATLAB 6 e executados em um computador com processador de 2,4GHz e 2GB de memória RAM. Cada instância do problema iRS/OS é representada por $K \times J \times R$, donde K é o número de pedidos, J o número de operações e R o número de máquinas.

Para os experimentos, o algoritmo proposto considera as funções objetivo: $f_1 =$ *Makespan* e $f_2 =$ Balanceamento de carga de trabalho, assim como o objetivo auxiliar $f_3 =$ tempo total de máquina parada. A metodologia proposta foi testada utilizando-se instâncias disponíveis em Moon *et al*. (2002).

Os resultados obtidos pelo algoritmo proposto foram comparados com o algoritmo genético abordado por Moon *et al*. (2004) e o moGA (ZHANG *ET AL*., 2006).

Foram realizados 2 experimentos considerando 50 e 100 soluções iniciais geradas aleatoriamente para cada instância. Os resultados do algoritmo proposto serão denotados como NBHA. Os resultados obtidos são presentados na Tabela 5.

Instância	Abordagem Moon *et al*. (2004)		moGA		NBHA (50)		NBHA (100)	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
5×21×5	1.500	$1{,}57\times10^4$	1.500	$1{,}57\times10^4$	1.490	2.440	1.490	2.440
					1.554	1.864	1.554	960
					1.635	960		
7×30×5	1.965	$1{,}49\times10^4$	1.894	$1{,}21\times10^4$	1.800	1.144	1.796	1216
					1.820	480	1.800	1.144
					1.950	376	1.820	480
					2.008	184	1.874	376
							1.886	264
							1.897	224
							2.008	184
							2.058	120
10×43×5	2.592	$1{,}94\times10^4$	2.487	$1{,}89\times10^4$	1.980	1.336	1.980	1.336
					1.990	1.224	1.990	1.224
					2.027	544	2.027	544
					2.055	424	2.055	424
					2.107	296	2.090	64
					2.260	200		
					2.390	120		

Tabela 5 - Resultados computacionais

Para todas as instâncias, observamos que o NBHA foi capaz de conseguir melhores resultados e soluções não-dominadas com 50 e 100 soluções iniciais, quando comparado com os algoritmos existentes.

O tempo médio de execução por solução gerada nas instâncias 5×21×5, 7×30×5 e 10×43×5 foi de 3seg, 17seg e 87seg respectivamente. Não foi possível comparar os tempos de execução do algoritmo proposto, pois não foram reportados os tempos computacionais dos algoritmos usados como referência.

6. Conclusões

Neste artigo foi considerado o problema multiobjetivo iRS/OS, para o qual é proposto um novo algoritmo heurístico hierárquico baseado no Método de Newton multiobjetivo para otimização contínua.

Experimentos numéricos com o algoritmo heurístico foram realizados para encontrar soluções não-dominadas. Os resultados mostram que o algoritmo proposto obtém boas soluções para as instâncias teste, considerando 50 e 100 soluções iniciais, quando comparada com algoritmos existentes para resolver as mesmas instâncias.

No entanto, quando maior seja o número operações e o número de máquinas será necessário incrementar o número de soluções iniciais do algoritmo para obter bons resultados. De fato, o algoritmo proposto, que não é uma heurística baseado em população, mas uma heurística *multi-start*, conseguiu soluções não-dominadas satisfatórias em todos as instâncias, mostrando seu potencial, com a vantagem de somente utilizar um parâmetro: o número de soluções iniciais.

Referências

Dayou, L., Pu, Y., Ji, Y. (2009), Development of a multiobjective GA for advanced planning and scheduling problem, *The International Journal of Advanced Manufacturing Technology*, 42, 974-992.

Fernández, M., Raupp, F. M. P. (2014), A Newton-based heuristic algorithm for multi-objective flexible job-shop scheduling problem, *Journal of Intelligent Manufacturing*, doi: 10.1007/s10845-014-0872-0.

Fliege, J., Drummond, L. M. G., Svaiter, B. (2009), Newton's method for multiobjective optimization, *SIAM Journal on Optimization*, 20, 602–626.

Kolisch, R., Hess, K. (2000), Efficient methods for scheduling make-to-order assemblies under resource, assembly area and part availability constraints, *International Journal of Production Research*, 38, 207-228.

Konak, A., Coit, D., Smith, A. (2006), Multi-objective optimization using genetic algorithms: a tutorial, *Reliability Engineering & System Safety*, 91, 992-1007.

Lee, Y. H., Yeong, C. S., Moon, C. (2002), Advanced planning and scheduling with outsourcing in manufacturing supply chain, *Computers & Industrial engineering*, 43, 351-374.

Moon, C., Kim, J., Gen, M. (2004), Advanced planning and scheduling based on precedence and resource constraint for e-plant chains, *International Journal of Production Research*, 42, 2941-2955.

Moon, C., Lee, M., Seo, Y., Lee, Y. H. (2002), Integrated machine tool selection and operation sequencing with capacity and precedence constraints using genetic algorithm, *Computers & Industrial Engineering*, 43, 605-621.

Tan, W., Khoshnevis, B. (2000), Integration of process planning and scheduling **-** a review, *Journal of Intelligent Manufacturing*, 11, 51-63.

Tan, W., Khoshnevis, B. (2004), A Linearized polynomial mixed integer programming model for the integration of process planning and scheduling, *Journal of Intelligent Manufacturing*, 15, 593-605.

Xia, W., Wu, Z. (2005), An effective hybrid optimization approach for muti-objective flexible job-shop scheduling problem, *Computers & Industrial Engineering*, 48, 409-425.

Yan, P., Liu, D., Yuan, D., Yu, J. (2007), Genetic Algorithm with Local Search for Advanced Planning and Scheduling, *Third International Conference on Natural Computation*, 3, 781-785.

Yang, J., Tang, W. (2009), Preference-based Adaptive Genetic Algorithm for Multiobjective Advanced Planning and Scheduling Problem, *IEEE International Conference on*, 1935-1940.

Zhang, H., Gen, M., Seo, Y. (2006), An effective coding approach for multiobjective integrated resource selection and operation sequences problem, *Journal of intelligent manufacturing*, 17, 385-397.