ALGORITMO VNS MULTI-OBJETIVO PARA UM PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM UMA MÁQUINA COM JANELAS DE ENTREGA

Rafael dos Santos Ottoni
Departamento de Informática - Universidade Federal de Viçosa
Campus da UFV, 36570-000, Viçosa, MG, Brasil
voviz.ottoni@gmail.com

José Elias Claudio Arroyo
Departamento de Informática - Universidade Federal de Viçosa
Campus da UFV, 36570-000 Viçosa, MG, Brasil
jarroyo@dpi.ufv.br

André Gustavo dos Santos
Departamento de Informática - Universidade Federal de Viçosa
Campus da UFV, 36570-000, Viçosa, MG, Brasil
andre@dpi.ufv.br

RESUMO

Neste artigo é abordado, do ponto de vista multi-objetivo, o problema de programação de tarefas em uma máquina com tempos de preparação de máquina dependentes da sequência e com janelas de entrega. Considera-se a minimização das penalidades totais por adiantamento e atraso e do tempo total de fluxo. Devido à complexidade do problema, é usado uma heurística eficiente baseada na técnica Variable Neighborhood Search (VNS) para determinar soluções não-dominadas de boa qualidade em tempo computacional razoável. É proposto um procedimento, de intensificação, com o objetivo de melhorar a eficiência de um algoritmo VNS multi-objetivo da literatura. O desempenho do algoritmo proposto é testado sobre um conjunto de problemas de médio e grande porte. Os resultados computacionais mostram que a técnica proposta é promissora para otimização multi-objetivo.

PALAVARAS CHAVE. Programação de tarefas, heurísticas, otimização multi-objetivo.

Área principal (Metaheurísticas)

ABSTRACT

In this paper, we examine the single machine scheduling problem with sequence dependent setup times and distinct due windows from a multiobjective point of view. We consider minimizing the total weighted earliness/tardiness and the total flowtime criteria. Due to the complexity of the problem, we use an efficient heuristic, based on the Variable Neighborhood Search (VNS), for searching good non-dominated solutions in reasonable computation times. We propose an intensification procedure to improve a multiobjective algorithm from the literature. The performance of the proposed algorithm is tested on a set of medium and large size problems instances. The computational results show that our approach is a promising technique for multiobjective optimization.

KEYWORDS. Job scheduling. Heuristics. Multiobjective optimization.

1. Introdução

O problema de programação de tarefas (*job scheduling*) em uma única máquina tem sido amplamente investigado durante as últimas décadas (Lenstra *et al*., 1977; Tan *et al*., 2000; Chen, 2005; Kuo e Yang, 2006). A maioria das contribuições considera um único critério de otimização, embora na prática a tomada de decisão esteja relacionada com vários critérios de desempenho (geralmente conflitantes). Os critérios ou objetivos mais comuns considerados na programação de tarefas são: minimização do tempo máximo de conclusão das tarefas (*makespan*), a minimização do *tempo total de fluxo*, a minimização do *atraso máximo* e a minimização do *atraso total*. A utilização destes objetivos é bem justificada na prática: a minimização do *makespan* implica na maximização da produção e utilização eficiente dos recursos, enquanto que a minimização do tempo total de fluxo está relacionada com a minimização do tempo de ciclo médio e redução do estoque em processamento (*Work-In-Process*). Os critérios relacionados com atraso estão relacionados com as datas de entrega das tarefas, tendo grande importância em sistemas de manufatura, pois penalidades podem ser pagas quando uma tarefa não for concluída dentro de sua data de entrega.

A maioria das pesquisas sobre problemas de sequenciamento de tarefas assumem que os tempo de preparação da máquina (*setup time*) são independentes da sequência de tarefas (Allahverdi *et al*., 2008). Em um sistema de produção, quando a máquina finaliza o processamento de uma tarefa e antes de executar uma outra tarefa, tempos de preparação são incorridos. A dimensão do *setup time* depende muitas vezes das semelhanças e requisitos tecnológicos das tarefas consecutivas. Normalmente um *setup time* elevado está associado a duas tarefas consecutivas que diferirem significativamente nos requisitos de processamento ou utilizarem diferentes tecnologias de processo.

O problema de programação de tarefas em uma única máquina (PT1M) com *setup times* dependentes da sequência e minimizando um único objetivo é um problema NP-difícil (Lenstra *et al*. 1977; Du e Leung, 1990; Baker e Scudder, 1990), o que torna um desafio a obtenção de soluções ótimas dentro de um limite de tempo razoável. Utilizando softwares de otimização "mono-objetivo" é possível obter soluções ótimas para instancias do problema de pequeno porte (número de tarefas < 20), em tempo aceitável.

A filosofia de produção *just-in-time* enfatiza a redução de antecipações e atrasos. Os produtos devem estar disponíveis no momento exato em que são requeridos (Liao e Cheng, 2007). Uma tarefa concluída antes da sua data de entrega (antecipação) poder gerar custos de manutenção de estoque, custos de armazenamento e possibilidades de deterioração. Em contrapartida, uma tarefa concluída com atraso pode gerar insatisfação dos clientes, penalidades do contrato, perda de vendas e reputação. Por estas razões, os objetivos envolvendo antecipação e atrasos da produção têm recebido uma atenção significativa recentemente (Wang e Yen, 2002; M'Hallah, 2007).

O problema PT1M com "janelas de entregas" e penalidades por antecipação e atraso é abordado na literatura como um problema mono-objetivo. Wang e Yen (2002) estudam o problema sem a consideração de *setup times*. Eles propõem um modelo de programação inteira, desenvolvem um algoritmo para o cálculo ótimo dos instantes de início/término das tarefas e utilizam um algoritmo baseado na heurística Busca Tabu para determinar soluções aproximadas. Souza *et al*. (2008) abordam o problema considerando *setup times* dependentes da sequência e propõem heurísticas baseadas em busca local, tais como GRASP (*Greedy Randomized Adaptive Search Procedure*), VND (*Variable Neighborhood Descent*) e Busca Tabu. Ribeiro *et al*. (2009) e Ribeiro *et al*. (2010) consideram o mesmo problema e utilizam algoritmos genéticos para sua resolução.

Este trabalho aborda o problema PT1M com janelas de entrega, penalidades por antecipação/atraso e tempos de *setup* dependentes da sequência e considera a minimização simultânea de dois objetivos: penalidade total por antecipação e atraso com relação às datas de entrega (f_1) das tarefas e tempo total de fluxo (f_2).

A qualidade de uma solução s de um problema de otimização bi-objetivo é avaliada por uma função vetorial $\boldsymbol{f} = (f_1, f_2)$: $X \rightarrow Z$, onde X é o espaço de decisão e Z o espaço objetivo ($Z \subseteq$

$\mathbf{R}^2$). Nesta classe de problema, geralmente, não existe uma solução que possui os valores mínimos dos objetivos (solução ideal). A comparação entre duas soluções s^1 e s^2 é baseada no conceito de dominância de Pareto. Diz-se que, s^1 *domina* s^2 se nenhum dos elementos de $\boldsymbol{f}(s^1)$ for maior que os correspondentes componentes em $\boldsymbol{f}(s^2)$, e pelo menos um dos componentes for estritamente menor, isto é, $f_1(s^1) \leq f_1(s^2)$, $f_2(s^1) \leq f_2(s^2)$ e $f_1(s^1) < f_1(s^2)$ ou $f_2(s^1) < f_2(s^2)$. Uma solução é chamada de Pareto-ótima se ela não é dominada por nenhuma solução do espaço de soluções factíveis. Portanto, a solução de um problema de otimização multi-objetivo (em particular, bi-objetivo) é um conjunto formado pelas soluções Pareto-ótimas (fronteira Pareto-ótima). A partir deste conjunto, o *decision maker* pode escolher a melhor solução de compromisso.

Os métodos mais usados para resolver problemas de otimização combinatória multi-objetivo são as metaheurísticas (Jones *et al.*, 2002; Gandibleux e Ehrgott, 2005). Métodos metaheurísticos foram concebidos originalmente para a otimização mono-objetivo e o sucesso alcançado na aplicação a uma grande variedade de problemas tem estimulado pesquisadores a adaptar estes métodos para problemas de otimização combinatória multi-objetivo. Revisões de pesquisas sobre metaheurísticas multi-objetivo revelam que a maioria dos artigos da literatura utilizam Algoritmos Genéticos (AGs) como a primeira metaheurística, seguido por *Simulated Annealing* e Busca Tabu (Jones *et al.*, 2002). A utilização de outras metaheurísticas, tais como GRASP, *Iterated Local Search* (ILS), *Variable Neighborhood Search* (VNS) e Colônia de Formigas é escassa. Recentemente estas metaheurísticas foram aplicadas para resolver alguns problemas de otimização combinatória multi-objetivo: GRASP (Arroyo *et al.*, 2008), ILS (Geiger 2009), VNS (Geiger, 2004; Liang e Lo, 2010) e Colônia de Formigas (Yagmahan e Yenisey, 2008).

Neste trabalho aplicas-se a metaheurística VNS para resolver o problema PT1M bi-objetivo. Propõe-se uma técnica de intensificação para melhorar o desempenho o algoritmo VNS multi-objetivo proposto por Geiger (2004). Resultados de testes computacionais demonstram a eficácia e robustez da técnica proposta.

2. Definição do Problema

O problema abordado neste trabalho é descrito da seguinte maneira. Existe n tarefas que devem ser processadas em uma única máquina que está continuamente disponível. Esta máquina pode processar apenas uma tarefa de cada vez. As n tarefas estão disponíveis para o processamento no tempo zero. Para cada tarefa j são conhecidos, o tempo de processamento p_j, uma janela de entrega $[de_j, dt_j]$ e as penalidades por antecipação (α_j) e atraso (β_j). Entre o processamento de duas tarefas consecutivas i e j é considerado um tempo de preparação (*setup*) da máquina s_{ij}. Neste problema é permitida a ocorrência de tempos de ocioso da máquina, ou seja, em alguns períodos de tempo a máquina pode ficar parada (sem executar nenhuma tarefa) até o início da próxima tarefa.

Se uma tarefa j finaliza dentro da sua janela de entrega $[de_j , dt_j]$ (ou seja, o tempo de conclusão C_j pertence ao intervalo $[de_j , dt_j]$), então a tarefa j não tem atraso ($T_j = 0$) e nem adiantamento ($E_j = 0$). Caso contrário incorrerá sobre a mesma penalidades por antecipação ($\alpha_i E_i$) ou atraso ($\beta_i T_i$), onde $E_j = \max\{de_j - C_j, 0\}$ e $T_j = \max\{C_j - dt_j, 0\}$.

O objetivo do problema é determinar uma sequência de tarefas $\boldsymbol{s} = (i_1,...,i_n)$ que minimize simultaneamente a penalidade total por antecipação e atraso ($f_1(\boldsymbol{s})$) e o tempo total de fluxo ($f_2(\boldsymbol{s})$). As funções f_1 e f_2 são calculados da seguinte maneira:

$$f_1(\boldsymbol{s}) = \sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j), \quad (1)$$

$$f_2(\boldsymbol{s}) = \sum_{j=1}^{n} C_j \quad (2)$$

Dada uma sequência (solução) $\boldsymbol{s}$, para calcular os objetivos $f_1(\boldsymbol{s})$ e $f_2(\boldsymbol{s})$ é necessário calcular os tempos de finalização C_j (*completion times*) das tarefas de $\boldsymbol{s}$. Os tempos de finalização são calculados utilizando o algoritmo ótimo, de tempo polinomial, proposto por Wang e Yen (2002). Já que são permitidos tempos ociosos na máquina, o algoritmo consiste em escalonar as

tarefas de tal maneira que finalizem o mais próximo possível da suas janelas de entrega.

Geralmente, o problema não possui uma única solução que minimize simultaneamente os dois objetivos. Na Tabela 1, apresentam-se os dados de entrada para uma instância do problema com 5 tarefas. Nas Figuras 1(a) e 1(b) são ilustrados os escalonamento das sequências de tarefas (1, 5, 3, 4, 2) e (5, 4, 1, 2, 3), respectivamente. Os valores dos objetivos para a primeira sequência são $f_1 = 0$ e $f_2 = 360$ e a para segunda sequência $f_1 = 580$ e $f_2 = 138$. Note que, no escalonamento da primeira sequência, existem tempos ociosos na máquina. Na Tabela 2 é apresentado um exemplo de um conjunto com 11 soluções não-dominadas para o problema da Tabela 1.

Tabela 1. Dados de entrada para uma instância do problema com 5 tarefas.

Parâmetros	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5
Tempo de processamento (p_j)	9	15	8	12	5
Data mais cedo (de_j)	15	150	22	140	21
Data mais tarde (dt_j)	25	170	30	180	22
Penalidade por antecipação (α_j)	3	2	4	1	5
Penalidade por atraso (β_j)	7	6	8	4	10

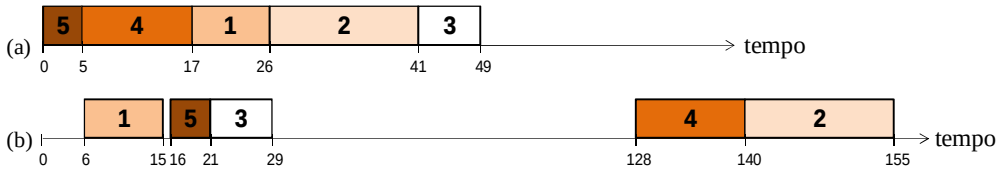


Figura 1. Exemplos de escalonamento de tarefas em uma máquina.

Tabela 2. Exemplo de soluções não-dominadas.

Sequência	f_1	f_2
5 4 1 2 3	580	138
1 4 5 2 3	547	146
2 5 3 4 1	542	157
4 5 3 2 1	532	163
5 2 3 1 4	424	230
5 4 3 1 2	266	231
5 4 1 3 2	242	232
4 5 3 1 2	211	238
4 5 1 3 2	187	239
1 3 5 4 2	53	358
1 5 3 4 2	0	360

3. Variable Neighborhood Search Aplicado ao Problema Programação de Tarefas

VNS, proposta originalmente por Mladenovic e Hansen (1997), é uma meta-heurística baseado em um princípio simples: mudança sistemática da estrutura de vizinhança no processo de busca local. Esta meta-heurística tem mostrado ser bastante eficiente na solução de problemas de otimização mono-objetivo, tais como o problema do caixeiro viajante e problemas de programação de tarefas (Hansen e Mladenovic, 2003).

Uma das primeiras aplicações da meta-heurística VNS para otimização multi-objetivo foi proposto por Geiger (2004). Geiger aplica seu algoritmo, denominado *MOVNS*, para resolver problemas de programação de tarefas *flowshop* bi-ojetivo. O algoritmo *MOVNS* de Geiger, recentemente, foi aplicado para resolver outros problemas de otimização multiobjetivo, tais como o problema de programação de tarefas em máquinas paralelas com mais de um ojetivo (Liang *et al.*, 2009) e o problema de alocação de redundância bi-objetivo (Liang e Lo, 2010).

No algoritmo *MOVNS*, a cada iteração, a estrutura de vizinhança e a solução a ser explorada (solução base) são escolhidas de forma aleatória. A solução base é selecionada do conjunto *D* de soluções não dominadas (aproximação do conjunto de soluções Pareto-ótimas). Esta solução é marcada, como visitada, para não ser selecionada nas próximas iterações, caso ela

permaneça no conjunto *D*. Se todas as soluções do conjunto *D* já estiverem selecionadas e a condição de parada do algoritmo ainda não for satisfeita, então todas as soluções são desmarcadas. A cada iteração do algoritmo, o conjunto *D* é atualizado com as soluções vizinhas geradas a partir da solução base.

Neste trabalho é usado o algoritmo *MOVNS* de Geiger para resolver o problema PT1M com penalidades por antecipação e atraso e fluxo total. Além disso, é proposto um procedimento de intensificação para melhorar o desempenho do algoritmo *MOVNS*. Este procedimento consiste em remover *d* tarefas de uma solução vizinha não-dominanda e em seguida reconstruir a solução (inserir as tarefas removidas) baseado numa técnica de enumeração parcial similar à heurística NEH (Nawas *et al.*, 1983).

Na Figura 2 mostra-se o pseudocódigo do algoritmo *MOVNS* implementado neste trabalho. O pseudocódigo do algoritmo *MOVNS* com intensificação (denominado *MOVNS_I*) é apresentado na Figura 3. O algoritmo do procedimento de intensificação é mostrado na Figura 4.

```
Algoritmo: MOVNS (t)
01 {s^1, s^2, s^3} := Construir 3 soluções (sequências) utilizando diferentes regras de prioridade;
02 D := conjunto de soluções não-dominadas de {s^1, s^2, s^3};
03 Enquanto Condição_Parada = falso faça
04       Selecione uma solução não visitada s de D (solução base);
05       Marque s como solução visitada;
06       Selecione aleatoriamente uma estrutura de vizinhança N_i do conjunto {N_1,...,N_t};
07       Determine aleatoriamente uma solução vizinha s'∈ N_i (s ); //shaking
08       Para cada vizinho s''∈N_i (s' ) faça
09           D := conjunto de soluções não-dominadas de D ∪ {s'' };
10       Fim-Para
11       Se todas as soluções de D estão marcadas como visitadas então
12            Remova as marcas das soluções (marcar como não-visitadas);
13 Fim-Enquanto
14 Retorne o conjunto D;
15 End MOVNS;
```

Figura 2. Pseudocódigo do algoritmo *MOVNS*

```
Algoritmo: MOVNS_I (t, d )
01 {s^1, s^2, s^3} := Construir 3 soluções (sequências) utilizando diferentes regras de prioridade;
02 D := conjunto de soluções não-dominadas de {s^1, s^2, s^3};
03 Enquanto Condição_Parada = falso faça
04       Selecione uma solução não visitada s de D (solução base);
05       Marque s como solução visitada;
06       Selecione aleatoriamente uma estrutura de vizinhança N_i do conjunto {N_1,...,N_t};
07       Determine aleatoriamente uma solução vizinha s'∈ N_i (s ); //shaking
08       D_1 := ∅;
09       Para cada vizinho s''∈N_i (s' ) faça
10           D_1 := conjunto de soluções não-dominadas de D_1 ∪ {s'' };
11       Fim-Para
12       Selecione aleatoriamente uma solução s do conjunto D_1;
13       D_2 := Intensificação(s, d); //Veja Figura 4.
14       D := conjunto soluções não-dominadas de D ∪ D_1 ∪ D_2;
15       Se todas as soluções de D estão marcadas como visitadas então
16            Remova as marcas das soluções (marcar como não-visitadas);
17 Fim-Enquanto
18 Retorne o conjunto D;
19 End MOVNS_I;
```

Figura 3. Pseudocódigo do algoritmo *MOVNS_I*.

O algoritmo *MOVNS_I* tem como entrada dois parâmetros, *t* o número de estruturas de vizinhanças e *d* o número de tarefas a serem removidas. O algoritmo retorna como saída o

conjunto D das soluções não-dominadas encontradas no processo da busca. O critério de parada dos algoritmos *MOVNS* e *MOVNS_I* é baseado em tempo computacional de execução.

Algoritmo: *Intensificação* ($\boldsymbol{s}$, d)
01 $D_2 := \emptyset$;
02 Gere aleatoriamente pesos w_1 e w_2, tal que $w_1 + w_2 = 1$;
03 Remova aleatoriamente d tarefas de $\boldsymbol{s}$ obtendo a sequência parcial $\boldsymbol{s}_p$ com n-d tarefas
04 Armazene as tarefas removidas em $\boldsymbol{s}_R$;
05 **Para** $i := 1,...,d$ **faça**
06 Inserir a tarefa $\boldsymbol{s}_R(i)$ em todas as posições de $\boldsymbol{s}_p$ gerando um conjunto S de (n–d+i) sequências parciais;
07 Avalie as sequências parciais $\boldsymbol{s}' \in S$; //*calcule* $f_1(\boldsymbol{s}')$ *e* $f_2(\boldsymbol{s}')$
08 **Se** $i = d$ **então** $D_2 :=$ conjunto de soluções não-dominadas de $D_2 \cup S$;
09 **Senão** Determine a melhor solução $\boldsymbol{s}_p$ de S (com relação a função $f_w = w_1 f_1 + w_2 f_2$);
10 **Fim-Para**
11 **Retorne** D_2;
12 **Fim Intensificação**;

Figura 4. Pseudocódigo da técnica de intensificação.

Nas seguintes subseções são descritas as etapas principais dos algoritmos *MOVNS* e *MOVNS_I*.

3.1 Determinação de soluções iniciais

Nos dois algoritmos VNS, inicialmente são geradas três soluções (ou sequências), denominadas $\boldsymbol{s}^1$, $\boldsymbol{s}^2$ e $\boldsymbol{s}^3$. As duas primeiras são determinadas usando a regra EDD (*Earliest Due Date*). Em $\boldsymbol{s}^1$ as tarefas são sequênciadas em ordem crescente de suas datas mais cedo de_j e em $\boldsymbol{s}^2$ as tarefas são ordenadas de acordo com suas datas mais tarde dt_j. Em $\boldsymbol{s}^3$ as tarefas são ordenadas de acordo com a regra SPT (*Shortest Processing Time*), ou seja, as tarefas são sequênciadas em ordem crescente do tempo de processamento das tarefas. Note que as soluções $\boldsymbol{s}^1$, $\boldsymbol{s}^2$ e $\boldsymbol{s}^3$ são construídas tentando minimizar o adiantamento, atraso e tempo de fluxo, respectivamente.

As três soluções geradas são avaliadas e as soluções não-dominadas são armazenadas no conjunto D.

3.2 Estruturas de vizinhança e busca local

Os algoritmos *MOVNS* e *MOVNS_I* usam 2 estruturas de vizinhança (t=2): inserção e troca. Estas vizinhanças são frequentemente utilizadas nos problemas de programação de tarefas. Para uma sequência $\boldsymbol{s} = (i_1,...,i_n)$, as vizinhanças são definidas da seguinte maneira:

- *Vizinhança inserção* (N_1): um vizinho de $\boldsymbol{s}$ é gerado inserindo a tarefa i_q, $1 \le q \le n$, em outra posição k da sequência, $k \ne q$. Se $k<q$, então $k \ne q$-1. A vizinhança $N_1(\boldsymbol{s})$ possui tamanho $(n\text{-}1)^2$.
- *Vizinhança troca* (N_2): um vizinho de $\boldsymbol{s}$ é gerado trocando as tarefas i_q e i_p na sequência, $1 \le q \le n, 1 \le p \le n, q \ne p$. A vizinhança $N_2(\boldsymbol{s})$, possui $n(n\text{-}1)/2$ soluções vizinhas.

A cada iteração do algoritmo VNS, uma estrutura de vizinhança N_i é selecionada aleatoriamente. A busca local inicia a partir de uma solução base $\boldsymbol{s}$ escolhida aleatoriamente do conjunto D (conjunto de soluções não-dominadas encontradas até o momento). Esta solução base é perturbada selecionando-se aleatoriamente uma solução vizinha $\boldsymbol{s}'$ de $N_i(\boldsymbol{s})$. Em seguida, todas as soluções vizinhas de $\boldsymbol{s}'$ são geradas, ou seja, todas as soluções em $N_i(\boldsymbol{s}')$ são calculadas.

No algoritmo *MOVNS*, o conjunto D é atualizado com as soluções de $N_i(\boldsymbol{s}')$. No algoritmo *MOVNS_I*, são determinadas as soluções não-dominadas de $N_i(\boldsymbol{s}')$ e armazenadas no conjunto D_1. O procedimento de intensificação inicia com uma solução não-dominada escolhida do conjunto D_1.

3.3 Procedimento de intensificação

O procedimento de intensificação é baseado na heurística de enumeração parcial proposto por Ruiz and Stützle, (2008). Cada vez que o algoritmo de intensificação é executado, um par de pesos w_1 e w_2 são gerados, tal que $w_1+w_2 = 1$. A intensificação é guiada pela função linear ponderada $f_w = w_1 f_1 + w_2 f_2$. Dessa maneira, a cada iteração são exploradas diferentes direções de busca definidas pelos pesos.

O procedimento de intensificação é composto de duas etapas: *destruição* e *reconstrução*. Na etapa de *destruição*, d tarefas são removidas da solução $\boldsymbol{s}$ ($\boldsymbol{s}$ é escolhida aleatoriamente do conjunto de soluções vizinhas não-dominadas D_1) e obtendo uma solução parcial $\boldsymbol{s}_p$. As tarefas removidas são armazenadas em $\boldsymbol{s}_R$ ($\boldsymbol{s}_R(i)$, i=1,...,d, são as tarefas removidas). A etapa de *reconstrução* possui d passos. No primeiro passo, (n-d+1) soluções parciais são construídas inserindo-se a tarefa $s_R(1)$ em todas as possíveis posições de $\boldsymbol{s}_p$. As (n-d+1) soluções parciais são avaliadas através da função ponderada f_w e a "melhor" solução (a de menor f_w) é selecionada. Esta solução selecionada substitui a solução $\boldsymbol{s}_p$. Os passos i = 2,…,d são similares. Note que, no passo $i = d$, n soluções completas são geradas. Destas n soluções, o conjunto das soluções não-dominadas é determinado, este conjunto é denominado D_2. O procedimento de intensificação retorna o conjunto D_2. Na Figura 5 ilustra-se um exemplo das etapas de *destruição* e *reconstrução* do procedimento de intensificação.

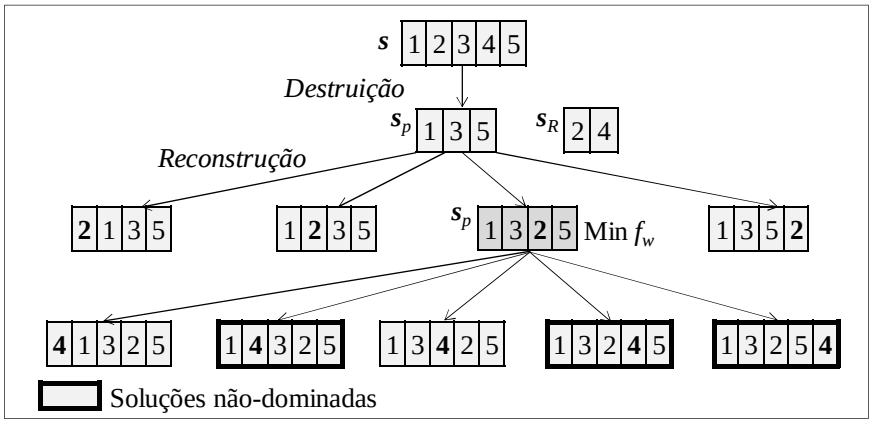


Figura 5. Procedimento de Intensificação

4. Resultados Computacionais

Neste trabalho analisa-se o desempenho do procedimento de intensificação utilizado no algoritmo *MOVNS* proposto por Geiger (2004). Para tanto, compara-se a qualidade das soluções obtidas pelos algoritmos *MOVNS* e *MOVNS_I*.

Os algoritmos foram codificados na linguagem C++ e executados em um microcomputador Intel Core Quad com 2.4GHz e 4GB de memória RAM. Nos dois algoritmos, é usada a mesma condição de parada. Foi usado um tempo máximo de CPU definido por 3n segundos. Este tempo de execução cresce linearmente com o tamanho do problema (número de tarefas n). Critérios de parada baseados em tempo de CPU são muito usados na literatura para a comparação de algoritmos heurísticos (Ruiz e Stützle, 2008).

No algoritmo *MOVNS_I*, para o parâmetro d, foram testados os seguintes valores: 2, 3, 4, 5, 6, 7, 8. Os melhores resultados foram obtidos para d = 4.

Já que os algoritmos analisados utilizam diversas escolhas aleatórias, os dois algoritmos foram executados 5 vezes com sementes diferentes. A partir das soluções obtidas nas 5 execuções de um algoritmo, determina-se o conjunto das soluções não dominadas.

4.1 Problemas testes

Os algoritmos *MOVNS* e *MOVNS_I* são testados sobre um conjunto de 72 instâncias do

problemas. O tamanho das instância são n = 20, 30, 40, 50, 75 e 100. As instâncias do problema foram gerados de acordo com os critérios usados por Wang e Yen (2002) e Ribeiro *et al.* (2009). Para cada tarefa j, o tempo de processamento p_j e a penalidade por atraso β_j são gerados aleatoriamente (com distribuição uniforme) nos intervalos [1, 100] e [20, 100], respectivamente. Na maioria dos casos reais, o atraso da produção é menos desejável do que a sua antecipação. Baseado nesta suposição, a penalidade por adiantamento α_j é k vezes a penalidade por atraso, sendo k uma variável aleatória e uniforme entre 0 e 1. Os centros das janelas de entrega $[de_j, dt_j]$ são uniformemente distribuídos dentro do intervalo $[1\text{-}T\text{-}RDD/2)\times TP,\ (1\text{-}T\text{-}RDD/2)\times TP]$, onde TP é o tempo total de processamento de todas as tarefas, T é o fator de atraso e RDD é a variação relativa da janela de entrega. Os valores considerados para T foram 0,1; 0,2; 0,3 e 0,4 e os valores considerados para RDD foram 0,8; 1,0 e 1,2. Os tamanhos das janelas de entrega são uniformemente distribuídas no intervalo $[1, TP/n]$. Os tempos de setup s_{ij} são gerados uniformemente dentro do intervalo [0,50]. Como em (Ribeiro *et al.*, 2009), utilizou-se tempos de preparação simétricos, ou seja, $s_{ij} = s_{ji}$.

Como T, RDD e n possuem, respectivamente, 4, 3 e 6 valores diferentes, foram gerados um total de 4×3×6 = 72 instâncias do problema (12 instâncias para cada tamanho n).

4.2 Medidas de avaliação de desempenho

Para uma determinada instância do problema, sejam, D_1 e D_2 os conjuntos de soluções não-dominadas encontradas pelos algoritmos *MOVNS* e *MOVNS_I* (nas 5 execuções), respectivamente. A partir dos conjuntos D_1 e D_2 determina-se o conjunto de soluções não-dominadas de referência denotado por *Ref* ($Ref = \{s \in D_1 \cup D_2;\ s$ é uma solução não-dominada$\}$).

Para avaliar a qualidade das soluções não-dominadas, são utilizadas três medidas (métricas): medida de cardinalidade, medida de distância e área de dominância (hipervolume).

Medida de cardinalidade: mede a quantidade de soluções de referência geradas por cada algoritmo, ou seja, são calculados os números $|Ref \cap D_1|$ e $|Ref \cap D_2|$. Note que, quanto maior o valor de $|Ref \cap D_i|$, melhor será a qualidade do respectivo algoritmo. Esta medida não avalia a distribuição das soluções sobre a fronteira Pareto-ótima.

Medida de distância: esta métrica mede a proximidade das soluções dos conjuntos D_1 e D_2 com relação às soluções do conjunto *Ref*. Além disso, mede a distribuição das soluções nos conjuntos D_1 e D_2. As seguintes fórmulas são usadas para calcular as distâncias média e máxima das soluções do conjunto D_i com relação às soluções do conjunto *Ref*:

$$D_{med}(D_i) = 100 \times \frac{1}{|Ref|} \sum_{s \in Ref} \min_{s' \in D_i} d(\boldsymbol{s}, \boldsymbol{s}') \qquad (3)$$

$$D_{max}(D_i) = \max_{s \in Ref} \{ \min_{s' \in D_i} d(\boldsymbol{s}, \boldsymbol{s}') \} \qquad (4)$$

onde, $d(\boldsymbol{s}, \boldsymbol{s}') = \max\{(f_1(\boldsymbol{s}) - f_1(\boldsymbol{s}'))/\Delta_1\ ,\ (f_2(\boldsymbol{s}) - f_2(\boldsymbol{s}'))/\Delta_2\}$ e Δ_j é diferença entre o maior e menor valor do objetivo f_j considerando as soluções do conjunto *Ref*. As medidas D_{med} e D_{max} são bastante usadas na avaliação de algoritmos de otimização multi-objetivo (Czyzak e Jaszkiewicz, 1998; Ishibuchi *et al.*, 2003; Framinan e Leisten, 2008).

Área de dominância: esta medida, denotada por H, foi proposta por Zitzler *et al.* (2003). Para o caso de mais de 2 objetivos, esta medida é chamada de hipervolume. Ela mede a área coberta ou dominada pelas soluções obtidas por um algoritmo. Para o caso da minimização de dois objetivos, um ponto de referencia (x, y) é usado para limitar essa área, onde x e y são limitantes superiores dos objetivos f_1 e f_2, respectivamente. Uma maior área de dominância indica que as soluções obtidas pelo algoritmo convergem melhor e geram uma boa cobertura da fronteira Pareto-ótima. Na Figura 6, ilustram-se as áreas de dominância de dois conjuntos de pontos, A e B. Neste exemplo gráfico, observa-se que $H(A) > H(B)$, o que indica que os pontos do conjunto A são "melhores" com relação a os pontos do conjunto B.

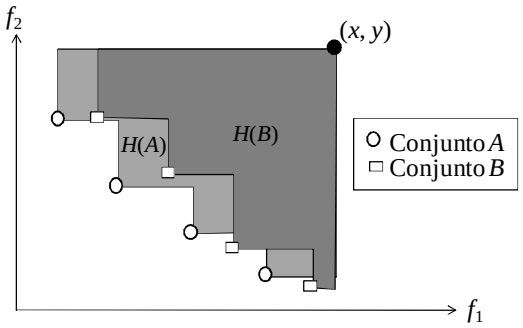


Figura 6. Exemplo de áreas de dominância para dois conjuntos de pontos.

4.3 Resultados Obtidos

Na Tabela 3 são comparados os algoritmos *MOVNS* e *MOVNS_I* utilizando a medida de cardinalidade. Para cada grupo de instâncias com *n* tarefas, são exibidos o número total de pontos nos conjuntos *Ref*, D_1, D_2, $Ref \cap D_1$ e $Ref \cap D_2$. Mediante análise da Tabela 3, nota-se que, os números de soluções obtidas por cada algoritmo (ou seja, $|D_1|$ e $|D_2|$) são relativamente próximos. Para todos os grupos de instâncias, o algoritmo *MOVNS_I* determina um número maior de soluções de referência, ou seja, $|Ref \cap D_2| > |Ref \cap D_1|$. Para as 75 instâncias resolvidas, 4098 soluções de referência foram obtidas, das quais 1072 e 3366 soluções foram determinadas, respectivamente, pelos algoritmos *MOVNS* e *MOVNS_I*. Baseado na medida de cardinalidade, o algoritmo *MOVNS_I* é superior ao *MOVNS*.

Tabela 3. Número de soluções obtidas pelos algoritmos

n		***MOVNS***		***MOVNS_I***	
	$\|Ref\|$	$\|D_1\|$	$\|Ref \cap D_1\|$	$\|D_2\|$	$\|Ref \cap D_2\|$
20	1025	996	529	987	836
30	992	836	245	917	747
40	793	676	207	718	586
50	656	558	54	635	602
75	429	460	25	406	404
100	203	152	12	192	191
Total de soluções	4098	3678	1072	3855	3366

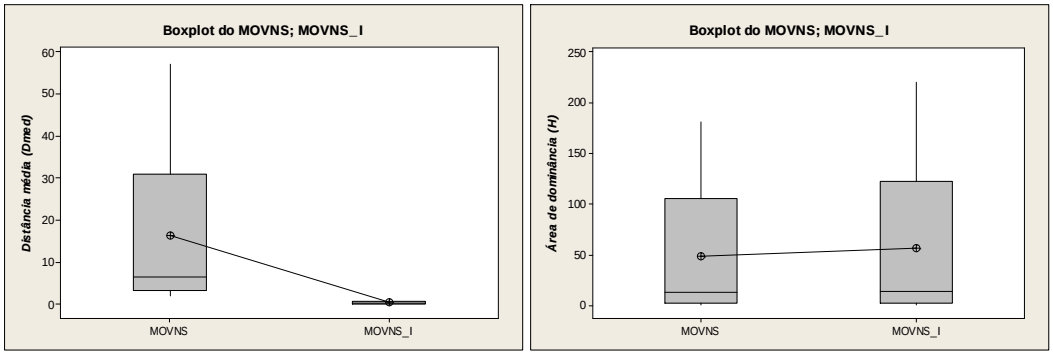
Tabela 4. Desempenho dos algoritmos: Medida de distância e área de dominância

n	***MOVNS***			***MOVNS_I***		
	$D_{med}(D_1)$	$D_{max}(D_1)$	$H(D_1)\times 10^{10}$	$D_{med}(D_2)$	$D_{max}(D_2)$	$H(D_2)\times 10^{10}$
20	5,31	14,22	0,752	0,21	2,95	0,789
30	2,04	6,58	3,388	0,59	5,86	3,427
40	3,90	10,22	9,005	1,25	7,82	9,173
50	7,74	17,66	18,233	0,24	3,10	19,199
75	22,35	33,76	80,549	0,13	1,77	89,842
100	57,10	71,68	181,844	0,54	1,33	220,235
Média	16,40	25,68	48,962	0,49	3,80	57,111

Na Tabela 4, os resultados obtidos pelos algoritmos *MOVNS* e *MOVNS_I* são comparados utilizando as medidas de distância e área de dominância. Para cada grupo de instâncias e para cada algoritmo, na Tabela 4, apresentam-se as médias do D_{med}, D_{max} e *H*. Observa-se que, para todos as instâncias, o algoritmo *MOVNS_I* é bem melhor considerando a medida de distância. Na Figura 7(a) mostra-se o *boxplot* da variação das distância D_{med} para os

dois algoritmos. Nesta Figura observa-se que a variação da distância D_{med} para o algoritmo *MOVNS* é muito maior do que para o algoritmo *MOVNS_I*. Isto mostra que existe diferença significativa entre os algoritmos, ao considerar a medida de distância.

A comparação pela média da área de dominância (*H*) observa-se que as soluções obtidas pelo algoritmo *MOVNS* são próximas às soluções do algoritmo *MOVNS_I*, no entanto, um número maior de soluções do *MOVNS* são dominadas pelas soluções do segundo algoritmo. Na Figura 7(b) mostra-se o *boxplot* da variação da área de dominância para os dois algoritmos. Devido à grande variação da medida *H*, não se detecta diferença significativa entre os dois algoritmos.



(a) (b)

Figura 7. Comparação dos algoritmos através da distância média e área de dominância.

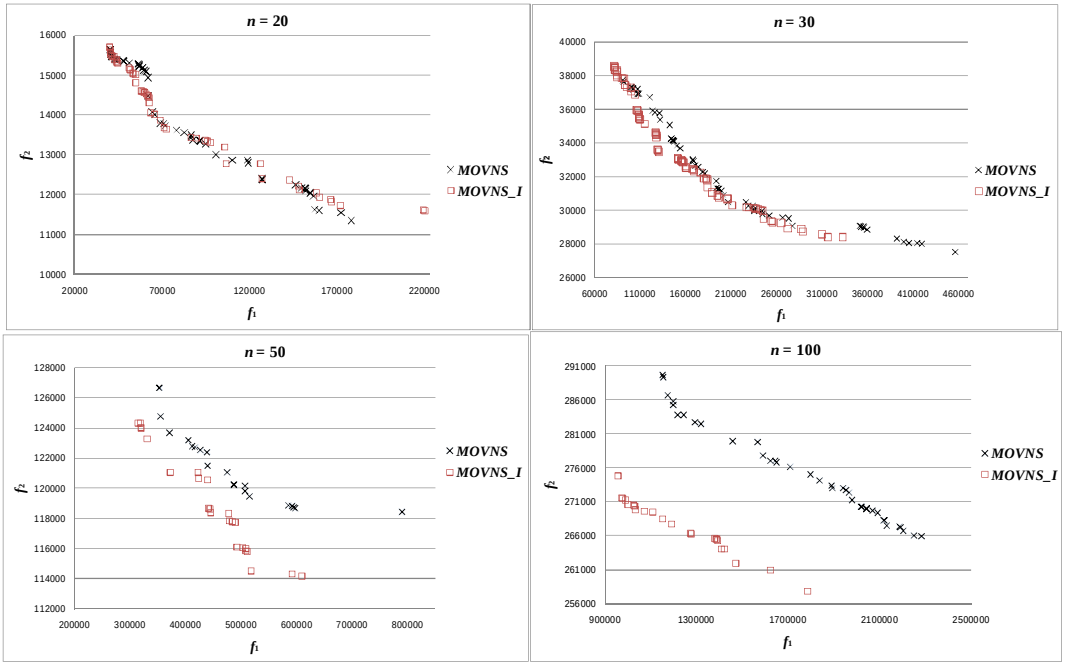


Figura 8. Pontos não-dominados determinados pelos algoritmos em quatro instâncias.

Na Figura 8, para quatro instância do problema de diferentes tamanhos, são mostradas as fronteiras de pontos determinados pelos algoritmos *MOVNS* e *MOVNS_I*. Observe que, para as instâncias com *n* = 20 e 30, os pontos determinados pelos algoritmos são bastante próximos. Já para as instâncias com *n* = 50 e 100, os pontos são mais distantes, sendo que os pontos encontrados pelo algoritmo *MOVNS-I* são melhores. Das fronteiras mostradas na Figura 8, pode-

se observar que os objetivos são conflitantes, um ponto com menor valor de f_1 possui o maior valor de f_2, e vice versa. Além disso, observa-se que, a faixa de variação dos objetivos são consideráveis. A variação de f_1 é sempre maior que a variação de f_2.

5. Conclusões

Neste artigo foram desenvolvidos dois algoritmos de otimização multi-objetivo para resolver o problema de programação de tarefas em uma máquina, com tempos de preparação de máquina, janelas de entrega e minimizando a penalidade total por adiantamento/atraso e o tempo total de fluxo. Os algoritmos são baseados na heurística VNS e o objetivo é determinar uma boa aproximação das soluções Pareto-ótimas. A ideia do primeiro algoritmo (denominado *MOVNS*) foi proposta na literatura e este algoritmo já foi aplicado para resolver diferentes problemas de otimização combinatória. Neste trabalho foi proposto um procedimento, chamado de intensificação, com o objetivo de melhorar o desempenho do algoritmo *MOVNS*. Este procedimento constrói novas soluções não-dominadas, a partir de uma solução "Pareto ótima local", baseado numa técnica de enumeração parcial. O desempenho do algoritmo *MOVNS* com intensificação (*MOVNS_I*) é testado em um conjunto de 75 instâncias do problema de médio e grande porte. Os resultados obtidos são avaliados utilizando três medidas de desempenho, número de soluções de referência obtidas, distância com relação às soluções de referencia e área de dominância. Mediante análise dos resultados conclui-se que o algoritmo proposto *MOVNS_I* determina, em geral, um número maior de soluções não-dominadas de referência de qualidade superior.

Agradecimentos

O presente trabalho foi financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e pela Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

Referências

Allahverdi, A., Ng, C.T., Cheng, T.C.E. e Kovalyov, M.Y. (2008), A survey of scheduling problems with setup times or costs, *Eur. Jour. of Operational Research*, 187, 985-1032.

Arroyo, J.E.C., Vieira, P.S. e Vianna, D.S. (2008), A GRASP algorithm for the multi-criteria minimum spanning tree problem, *Annals of Operations Research*, 159,125–133.

Baker, K.R. e Scudder, G.D. (1990), Sequencing with earliness and tardiness penalties: a review, *Operations Research*, 38,22-36.

Chen, W. J. (2005) Minimizing total flow time in the single-machine scheduling problem with periodic maintenance, *J. Oper. Res. Soc.*, 57, 410-415.

Czyzak, P. e Jaszkiewicz, A. (1998), Pareto simulated annealing - a metaheuristic technique for multiple objective optimization. *J Multi-Crit Decis Making*, 7 pp. 34-47.

Du, J. e Leung, J.Y.T. (1990), Minimizing total tardiness on one machine is NP-hard, *Mathematics of Operations Research*, 15, 483-495.

Framinan, J.M. e Leisten, R. (2008), A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. *OR Spectrum*, 30:787–804.

Gandibleux, X. e Ehrgott, M. (2005), 1984-2004, 20 years of multiobjective metaheuristics, *Lecture Notes in Computer Science*, 3410, 33–46.

Geiger, M.J. (2004), Randomized variable neighborhood search for multi objective optimization, *4th EU/ME: Design and Evaluation of Advanced Hybrid Meta-Heuristics* (2004), pp. 34-42.

Geiger, M.J. (2009), Improvements for multi-objective flow shop scheduling by Pareto Iterated Local Search, *Proceedings of the 8th Metaheuristics International Conference* (MIC 2009), pp. 195.1–195.10.

Hansen, P. e Mladenovic, N. (2003), Variable Neighborhood Search, *Kochenberger Handbook of metaheuristics, F. Glover Eds. Kluwer Academic*, pp.145–184.

Ishibuchi, H., Yoshida, T. e Murata, T. (2003), Balance between genetic local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Trans Evol Comput* 7(2): pp. 204-223.

Jones, D.F., Mirrazavi, S.K. e Tamiz, M. (2002), Multi-objective metaheuristics: An overview of the current state-of-art, *Eur. Jour. of Operational Research*, 137, 1-19.

Kuo, W-H. e Yang, D-L. (2006), Minimizing the makespan in a single machine scheduling problem with a time-based learning effect, *Information Processing Letters*, 97, 64-67.

Lenstra, J.K., Rinnooy Kan, A.H.G. e Brucker, P. (1977), Complexity of machine scheduling problems, *Annals of Discrete Mathematics*, pp. 343–362.

Liang, Y-C., Chen, H-L. A. e Tien C-Y. (2009), Variable Neighborhood Search for Multi-Objective Parallel Machine Scheduling Problems, *Proceeding of the 8th International Conference on Information and Management Sciences* (IMS 2009), China, pp. 519-522.

Liang, Y-C e Lo, M-H. (2010), Multi-objective redundancy allocation optimization using a variable neighborhood search algorithm, *Journal of Heuristics*, 16, 511-535.

Liao, C-J. e Cheng, C-Cg. (2007), A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date, *Comp Ind Eng.*, 52, 404-413.

M'Hallah, R. (2007), Minimizing total earliness and tardiness on a single machine using a hybrid heuristic, *Computers and Operations Research*, 34, 3126-3142.

Mladenovic, N. e Hansen, P. (1997), Variable Neighborhood Search, *Computers and Operations Research*, 24, 1097–1100.

Nawaz, M., Enscore, E.E. e Ham, I. (1993), A heuristic algorithm for the m-machine. n-job flowshop sequencing problem, *OMEGA*, 11, 91-95.

Ribeiro, F.F., Souza, S.R. e Souza, M.J.F. (2009), An adaptive genetic algorithm for solving the single machine scheduling problem with earliness and tardiness penalties. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. San Antonio, USA, 698-703.

Ribeiro, F.F., Souza, S.R., Souza, M.J.F. e Gomes, R.M. (2010), An adaptive Genetic Algorithm to Solve the Single Machine Scheduling Problem with Earliness and Tardiness. *Proceedings of the IEEE Congress on Evolutionary Computation*, 1-8.

Ruiz, R. e Stützle, T. (2008), An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187: 1143-1159.

Souza, M.J.F., Penna, P.H.V. e Gonçalves, F.A.C.A. (2008), GRASP, VND, busca tabu e reconexão por Caminhos para o problema de sequenciamento em uma máquina com tempos de preparação dependentes da seqüência da produção, janelas de entrega distintas e penalidades por antecipação e atraso da produção. *Anais do XL Simpósio Brasileiro de Pesquisa Operacional SBPO*, 1320-1331.

Tan, K.C., Narasimhan R., Rubin, P.A. e Ragatz, G.L. (2000), A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times, *OMEGA*, 28, 313-326.

Wang, G. e Yen, B.P. (2002), Tabu Search for Single Machine Scheduling with Distinct Due windows and Weighted Earliness/Tardiness Penalties, *Eur. Jour. of Operational Research*, 142, 129–146.

Yagmahan, B. e Yenisey, M.M. (2008), Ant colony optimization for multiobjective flow shop scheduling problem, *Comp Ind Eng.*, 54, 411–420.

Zitzler, E., Thiele, L., Laumanns, M. Foneseca, C.M. e Grunert F. V. (2003), Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117-132.