HEURÍSTICA BUSCA LOCAL ITERADA PARA O SEQUENCIAMENTO DE TAREFAS EM UMA MÁQUINA COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA FAMÍLIA

Vinícius Vilar Jacob, José Elias C. Arroyo, André Gustavo dos Santos
Departamento de Informática, Universidade Federal de Viçosa (UFV)
Campus Universitário, Viçosa – Minas Gerais – MG – Brasil – 36.570-000
vinicius.jacob@ufv.br, jarroyo@dpi.ufv.br, andre@dpi.ufv.br

RESUMO

Testee2 artigo aborda um problema de programação da produção em uma máquina com tempos de preparação (*setup*) dependente da sequência das famílias de tarefas. Neste problema as tarefas são agrupadas em famílias de acordo com características de similaridade e um tempo de preparação é necessário entre o processamento de duas tarefas de famílias distintas. O critério a ser minimizado é o atraso total das tarefas com relação a suas datas de entrega. O problema é classificado como NP-Difícil no senso comum, então, heurísticas eficientes são necessárias para obter soluções próximas da ótima em tempo computacional razoável. Neste trabalho é proposta uma heurística baseada em *Iterated Local Search* em que a perturbação é aplicada em níveis. O desempenho da heurística proposta é comparada com um Algoritmo Genético (AG) proposto na literatura para o mesmo problema. Depois de uma extensiva análise computacional e estatística conclui-se que a heurística proposta tem performance superior ao AG com respeito a qualidade das soluções encontradas.
PALAVRAS CHAVE. Sequenciamento em uma máquina, Tempo de preparação dependente das famílias, Atraso total, Busca local iterada.

Áreas Principais: MH - Metaheuristicas, OC - Otimização Combinatória.

ABSTRACT

This paper addresses the single machine job scheduling problem with sequence dependent family setup time. In this problem the jobs are categorized into families according to their similarity characteristics and setup times are required on each occasion when the machine switches from processing jobs in one family to jobs in another family. The criterion to be minimized is the total tardiness. The problem is classified as NP-hard in the ordinary sense, thus, efficient heuristics are needed to obtain near-optimal solutions in reasonable computational time. In this work we propose an Iterated Local Search (ILS) heuristic in which the perturbation is applied in levels. The performance of the proposed heuristic is compared with a Genetic Algorithm (GA) proposed in the literature for the same problem. After extensive computational and statistical analysis we can conclude that the proposed heuristic performs better than GA with respect to quality of found solutions.
KEYWORDS: Single machine scheduling, Family setup times, Total tardiness, Iterated local search.

Main areas: MH - Metaheuristics, OC - Combinatorial Optimization.

1 Introdução

Tamannaei e Rasti-Barzoki (2019) s,s,s, A programação ou sequenciamento da produção é um processo de tomadas de decisões que ocorre em sistemas de manufaturas. Problemas de programação da produção têm sido investigados desde meados dos anos 50 (Allahverdi *et al.*, 2008) e ainda na atualidade continuam sendo muito estudados. A importância de se estudar tais problemas provém principalmente por dois aspectos: o primeiro diz respeito a sua importância prática, com várias aplicações em diversos setores, como indústrias química, metalúrgica e têxteis. O segundo aspecto é sobre a dificuldade para se resolver a maioria dos problemas de sequenciamento, pois eles pertencem a classe NP-Difícil de problemas.

O problema da programação focado neste trabalho é descrito a seguir. Há um conjunto de n tarefas a serem processadas em uma máquina sem interrupção ou preempção. As tarefas são divididas em F famílias e estão disponíveis num tempo zero. Cada família l tem n_l tarefas, tal que $n_1 + n_2 + ... + n_l = n$. $f(j)$ denota a família da tarefa j. O tempo de processamento e a data de entrega da tarefa j são conhecidos e definidos por p_j e d_j, respectivamente. Há um tempo de preparação s_{jk} entre as tarefas j e k se a tarefa k é processada imediatamente depois da tarefa j. Se as tarefas j e k são da mesma família, $f(j) = f(k)$, não há nenhum tempo de preparação ($s_{jk} = 0$). O tempo de preparação é necessário, por exemplo, para o ajuste de ferramentas, limpeza, posicionamento de acessórios, inspeção de materiais, dentre outros. O tempo de preparação é dependente da sequência, isto é, $s_{jk} \neq s_{kj}$. O tempo de preparação para a primeira tarefa de uma sequência é zero. O problema consistem em encontrar a ordem de processamento das tarefas (sequência) de modo a minimizar a atraso total das tarefas com relação a suas datas de entregas.

O tempo de conclusão de uma tarefa j é definido como C_j. Se uma tarefa termina antes da sua data de entrega, não há atraso. Caso contrário, o atraso incorrido é dado por $T_j = max(C_j - d_j, 0)$. O atraso total das tarefas (função objetivo) é computado como:

$$f = \sum_{j=1}^{n} T_j \tag{1}$$

O problema é denotado por $1|ST_{sd,b}|\sum T_j$ seguindo a notação de três campos $\alpha|\beta|\gamma$ introduzida por Graham *et al.* (1979) e adaptada por Allahverdi *et al.* (2008), onde 1 denota o ambiente com uma única máquina, $ST_{sd,b}$ é a informação do tempo de preparação dependente da sequência dos lotes ou famílias e $\sum T_j$ é o critério do atraso total.

Uma vez que o problema de minimização de atraso total em uma máquina sem famílias é NP-Difícil (Du e Leung, 1990), segue que o problema $1|ST_{sd,b}|\sum T_j$ é pelo menos NP-Difícil no senso comum.

O critério de atraso total é muito importante em sistemas de manufaturas, porque podem existir diversos custos quando uma tarefa é entregue com atraso. Dentre estes podem ser citados: multas contratuais, perda de credibilidade resultando em alta probabilidade de se perder um cliente para alguma ou para todas as tarefas futuras, e danos na reputação da empresa que pode afastar outros clientes. Além disso, este critério é menos estudado e mais difícil de se trabalhar que outros critérios relacionados apenas ao tempo de conclusão.

Problemas de programação da produção que consideram explicitamente tempos de preparação são de grande importância em sistemas de manufatura. Uma extensa revisão de literatura para estes problemas, considerando diferentes ambientes de produção, foi realizado por Allahverdi *et al.* (2008).

Alguns estudos foram realizados sobre o problema de sequenciameto em uma máquina com tarefas agrupadas em famílias e tempos de preparação independente das

famílias ($ST_{si,b}$). Nos trabalhos realizados por Kacem (2006) e Schaller (2007) foram propostos algorítimos *branch-and-bound* e heurísticas para o problema $1|ST_{si,b}|\sum T_j$. Para a minimização de adiantamentos e atrasos ($\sum E_j + \sum T_j$), métodos exatos e heurísticos foram desenvolvidos por Schaller e Gupta (2008). Abordagens exatas também foram usadas por Pan e Su (1997) e Baker e Magazine (2000) para resolver o problema $1|ST_{si,b}|L_{max}$, onde L_{max} é o *lateness* máximo.

No entanto, há poucos artigos para o problema de programação da produção em uma máquina com tempos de preparação dependentes da sequência e das famílias ($1|ST_{sd,b}|*$). Van der Veen *et al.* (1998) abordou o problema $1|ST_{sd,b}|C_{max}$ para minimizar o tempo de conclusão máximo (C_{max}). Estes autores propuseram um algoritmo de tempo polinomial. Para o problema de minimização do tempo total de conclusão ($1|ST_{sd,b}|\sum C_j$), Karabati e Akkan (2005) propuseram um algoritmo *branch-and-bound*. No artigo de Jin *et al.* (2010) é abordado o problema $1|ST_{sd,b}|L_{max}$, no qual os autores usam uma heurística de busca tabu.

No melhor do nosso conhecimento, o único artigo que aborda o problema $1|ST_{sd,b}|\sum T_i$ foi apresentado por Chantaravarapan *et al.* (2003). Os autores propuseram um algoritmo genético hibrido (HGA). O HGA foi mais eficiente em minimizar o atraso total quando comparado com outros algoritmos heurísticos.

Neste artigo, é apresentado um algoritmo heurístico baseado na metaheurística *Iterated Local Search* (ILS) para obter soluções aproximadas de alta qualidade para o problema $1|ST_{sd,b}|\sum T_j$. ILS é uma metaheurística simples, robusta e efetiva que aplica repetidamente uma busca local a soluções obtidas ao perturbar soluções ótimas locais previamente visitadas. Na heurística proposta, a perturbação é aplicada em níveis, isto é, a perturbação é aumentada à medida que a busca local não consegue gerar novas soluções melhores.

O restante deste artigo está organizado como segue: na Seção 2 é descrito os métodos usados na heurística ILS proposta. Os experimentos computacionais são apresentados na Seção 3. Os resultados obtidos são analisados estatisticamente na última Seção. Finalmente, na Seção 4, estão as conclusões do trabalho.

2 Heurística Proposta

Neste trabalho é desenvolvido um algoritmo heurístico baseado na metaheurística *Iterated Local Search* (ILS) para resolver o problema $1|ST_{sd,b}|\sum T_j$. ILS (Lourenço *et al.*, 2003) é uma metaheurística simples e de aplicação geral que usa busca local em modificações da solução corrente. Quatro métodos básicos são necessários para derivar um algoritmo ILS: um método "Gerar-Solução-Inicial", que encontra uma solução inicial S, um método "Perturbação" que perturba a solução corrente S levando a alguma solução intermediária S_1, um método de "Busca Local" que melhora S_1 obtendo um ótimo local S_2, e um critério de aceitação que decide para qual solução a próxima perturbação é aplicada. O critério de aceitação define um compromisso entre diversificação e intensificação. Uma intensificação muito forte é alcançada se apenas soluções melhores que a corrente são aceitas (S_2 melhor que S). No outro extremo, há grande diversificação quando soluções são aceitas aleatoriamente, independente do valor da função objetivo. ILS é uma poderosa técnica de otimização, e muitos bons resultados foram obtidos para uma variedade de problemas de otimização combinatória, tais como o problema do caixeiro viajante (Martin e Otto, 1996), programação de tarefas no ambiente *job shop* (Lourenço, 1995) e no ambiente *flowshop* (Dong *et al.*, 2009).

Na heurística proposta, é utilizado controle automático da perturbação cuja intensidade é aumentada à medida que o procedimento de busca local não consegue gerar novas soluções melhores. Nas próximas subseções, serão descritos os métodos da heurística

ILS proposta.

2.1 Representação de uma Solução

Uma solução (sequenciamento) do problema de programação de tarefas é representado por uma permutação de n tarefas $S = \{j_1, j_2,..., j_n\}$, onde j_k é a k-ésima tarefa na sequência de processamento. Por exemplo, considere uma instância com $n = 7$ tarefas que é especificada na Tabela 1. A família, o tempo de processamento e a data de entrega de cada tarefa estão na Tabela 1a, e os tempos de preparação entre as famílias são mostrados na Tabela 1b. Para o sequenciamento $S = \{7, 1, 5, 4, 2, 6, 3\}$ os tempos de conclusão e atrasos das tarefas são ($C_7 = 2$, $C_1 = 5$, $C_5 = 10$, $C_4 = 12$, $C_2 = 14$, $C_6 = 19$, $C_3 = 23$) e ($T_7 = 0$, $T_1 = 3$, $T_5 = 2$,$T_4 = 1$, $T_2 = 7$, $T_6 = 4$, $T_3 = 5$), respectivamente. Logo, o atraso total é 22. Este sequenciamento é ilustrado na Figura 1. Note que há a formação de quatro lotes e três tempos de preparação são requeridos (não há tempo de preparação no início da sequência).

Tabela 1: Dados de entrada para uma instância.

Job i	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$f(i)$	1	2	1	2	2	1	2
d_i	2	7	18	11	8	15	3
p_i	1	2	4	2	4	3	2

(a) Família, data de entrega e tempo de processamento.

s_{ik}	**1**	**2**
1	0	1
2	2	0

(b) Tempo de preparação entre famílias.

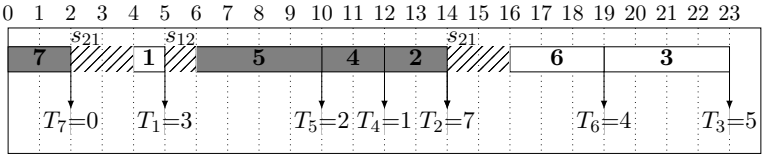

Figura 1: Sequenciamento $\mathbf{S} = \{\mathbf{7}, \mathbf{1}, \mathbf{5}, \mathbf{4}, \mathbf{2}, \mathbf{6}, \mathbf{3}\}$.

2.2 Construção da Solução Inicial

Para construir uma solução inicial para o ILS, foi utilizado um algoritmo baseado na heurística NEH (Nawaz *et al.*, 1983) adaptada para a minimização do atraso total. Nesta heurística primeiramente as tarefas são arranjadas em ordem não-decrescente das suas datas de entrega (regra *Earliest Due Date*) formando a lista ordenada $J = \{j_1, ...,j_n\}$ de tarefas. A primeira tarefa de J é inserida na sequência parcial S ($S = \{j_1\}$). Cada tarefa restante j_i de J ($i = 2,...,n$) é inserida em todas as posições possíveis da sequência S gerando i sequências parciais. Das i sequências, a melhor (em relação ao atraso total parcial) é selecionada. A heurística termina quando todas as tarefas da lista J são inseridas na sequência S.

2.3 Busca Local

A busca local é usada para melhorar a solução inicial S e as soluções obtidas pelo procedimento de perturbação. Busca local é um método que começa com uma solução S, e gera uma vizinhança que contém todas as soluções que são alcançadas através de movimentos individuais realizados na solução corrente. Desta vizinhança, a solução que é melhor que a solução corrente é selecionada. A solução escolhida torna-se a nova solução corrente e o processo continua até que um ótimo local seja alcançado.

Neste trabalho foi utilizado o procedimento de Busca Local proposto por Ruiz e Stutzle (2008) que é adaptado para sequenciamento em uma máquina e minimização do

atraso total. O procedimento de busca local é baseado na vizinhança por inserção. Esta vizinhança consiste em remover uma tarefa j da sua posição original e inserir ela nas $n-1$ posições restantes. Este movimento gera uma vizinhança de tamanho $(n-1)^2$. A solução corrente é substituída por uma solução vizinha que melhore o valor do atraso total. O procedimento é repetido enquanto há melhoria na solução corrente. O procedimento de busca local utilizado neste artigo é mostrado no Algoritmo 1. Para reduzir o tamanho da vizinhança,é utilizado o parâmetro γ que define a probabilidade para uma tarefa j ser inserida em todas as posições da solução corrente. Se $\gamma = 1.0$, a vizinhança inteira é avaliada assim como em Ruiz e Stutzle (2008). O algoritmo possui também o parâmetro FFI (*flag first improvement*) que define o critério de escolha da solução vizinha a ser explorada na próxima iteração. Note que se $FFI = 1$ (linha 14) na busca local é usada a estratégia de escolha do primeiro vizinho melhor (*First Improvement*). Caso contrário, o melhor da vizinhança (*Best Improvement*) é usado na próxima iteração da busca local.

Algoritmo 1: Busca_Local (S, γ, FFI)

```
    saída: Solução S melhorada
 1  início
 2      melhora := true;
 3      enquanto melhora faça
 4          melhora := false;
 5          J := {j1,...,jn} conjunto de tarefas ordenadas aleatoriamente;
 6          para i := 1 até n faça
 7              j := aleatoriamente selecionar uma tarefa de J;
 8              J := J − {j} ;
 9              se rand(0..1) ≤ γ então
10                  S' := melhor solução obtida ao inserir j em todas as posições de S;
11                  se f(S') < f(S) então
12                      S := S';
13                      melhora := true;
14                      se FFI = 1 então
15                          Vá para a linha 4 /*Primeiro aprimorante*/
16      retorna S
```

2.4 Perturbação

O objetivo do método de perturbação em uma heurística ILS é escapar de uma solução ótimo local. A perturbação de uma solução deve ser forte o bastante para sair do mínimo local corrente e habilitar à busca local para encontrar um novo, possivelmente melhor, mínimo local. Ao mesmo tempo, a perturbação deve ser fraca o bastante para manter boas características do mínimo local corrente (Gendreau e Potvin, 2010). Neste trabalho, foi utilizado um procedimento de perturbação similar ao proposto por Rego *et al.* (2012). Uma solução é perturbada realizando um número de trocas entre tarefas e é aplicado em níveis. Se a melhor solução não é melhorada pela busca local por um número de N iterações consecutivas, o nível da perturbação é incrementado, caso contrário o nível da perturbação retorna ao seu valor mais baixo.

O nível d da perturbação é um valor no intervalo $d \in [1, d_{max}]$, $d_{max} \leq (n/2 - 1)$. No nível d, $d+1$ trocas são feitas na solução. Desta maneira, no primeiro nível duas trocas são aplicadas, enquanto que no nível mais alto $n/2$ trocas são feitas.

Para perturbar uma solução S, uma posição j ($0 \leq j \leq n-2d-2$) é aleatoriamente escolhida de S. As trocas são feitas no subconjunto de tarefas consecutivas de S nas posições $j, j+1,...,j+2d+1$. Então, as trocas são aplicadas entre os pares das posições: $(j, j+2d+1)$, $(j+1, j+2d)$,...,$(j+d, j+d+1)$. Desta maneira, $d+1$ movimentos de trocas são realizados

em cada chamada do procedimento de perturbação.

2.5 Estrutura do algoritmo ILS com controle automático da pertubação

Uma descrição geral do pseudocódigo do algoritmo proposto com controle automático da perturbação (chamado ILS_Melhorado) é apresentado no Algoritmo 2. O algoritmo tem cinco parâmetros de entrada: o parâmetro N (que controla a atualização do nível da pertubação), d_{max} (o nível máximo da perturbação), γ (probabilidade de uma solução escolhida na busca local ter sua vizinhança avaliada), β (probabilidade de aceitação da solução melhorada mesmo que seja pior que a solução corrente) e FFI (que define qual o critério de seleção do vizinho será utilizado na busca local). Na linha 2 uma solução inicial é construída e é melhorada pelo procedimento *Busca Local* (linha 2). Na linha 4 o nível da perturbação é inicializado ($d = 1$). As iterações do algoritmo ILS são computadas nas linhas 7-22 até que o critério de parada seja satisfeito. Durante cada iteração, a solução corrente S é perturbada (linha 8) e melhorada pela *Busca Local*, obtendo uma solução S_2 (linha 9). Nas linhas 10-13, se a solução S_2 melhora a melhor solução obtida até o momento, o nível da perturbação é alterado para o seu menor valor ($d = 1$). Se a melhor solução não é melhorada durante N iterações consecutivas, o nível da perturbação é incrementado (veja linhas 15 e 17). O maior valor para o nível da pertubação é d_{max}. Nas linhas 18-22 é testado o critério de aceitação. Se S_2 é melhor que a solução corrente S então ela é aceita (S_2 substitui S), caso contrário ela pode ser aceita com uma pequena probabilidade β, de modo que haja um balanço entre diversificação e intensificação. A melhor solução encontrada durante todas as iterações é retornada pelo algoritmo (linha 23).

Algoritmo 2: ILS_Melhorado (N, d_{max}, γ, β, FFI)

```
    saída: Melhor solução
 1  início
 2      S := Gera_Solução_Inicial();
 3      S := Busca_Local(S, γ, FFI);
 4      d := 1;
 5      cont := 0;
 6      S* := S; //melhor solução
 7      enquanto critério de parada faça
 8          S1 := Perturbação(S, d);
 9          S2 := Busca_Local(S, γ, FFI);
10          se f(S2) < f(S*) então
11              S* := S2;
12              cont := 0;
13              d := 1;
14          senão
15              cont := cont + 1;
16              se cont mod N = 0 então
17                  d := min(d + 1, dmax);
18          se f(S2) < f(S) então
19              S := S2;
20          senão
21              se rand(0..1) ≤ β então
22                  S := S2;
23      retorna S*
```

3 Experimentos Computacionais

Neste trabalho, foi comparado o algoritmo ILS_Melhorado com o Algoritmo Genético Híbrido (HGA) proposto por Chantaravarapan *et al.* (2003) para resolver o

problema $1|ST_{sd,b}|\sum T_j$ abordado neste trabalho. O algoritmo HGA foi reimplementado seguindo o artigo original (Chantaravarapan *et al.*, 2003). Também é analisado a eficiência do controle automático da perturbação usado no algoritmo ILS; consequentemente foi comparado ILS_Melhorado com a versão básica que usa um nível de perturbação fixo (chamado ILS_Básico). Todos os algoritmos foram codificados em C++ e executados em uma máquina com processador Intel(R) Xeon(R) CPU X5650 @ 2.67GHz e 48 GB de RAM.

3.1 Instâncias de teste

As instâncias do problema $1|ST_{sd,b}|\sum T_j$ foram geradas aleatoriamente de acordo com o trabalho de Jin *et al.* (2010). Foram considerados problemas testes com o número de tarefas $n \in \{60,80,100\}$ e número de famílias $F \in \{2,3,4,5\}$. O tempo de processamento das tarefas (p_j) foram gerados aleatoriamente, com distribuição uniforme, sobre o intervalo $[1;99]$. Os tempos de preparação são uniformemente distribuídos em três classes de intervalos: **classe S** em $[11;20]$; **classe M** em $[51;100]$ e **classe L** em $[101;200]$. As datas de entrega são números inteiros no intervalo $[0; r\sum_{j=1}^{n} p_j]$, onde r é um parâmetro usado para controlar a amplitude da data de entrega, $r \in \{0{,}5; 1{,}5; 2{,}5; 3{,}5\}$. Combinando os parâmetros n, F, r e classes de tempo de preparação, tem-se 144 configurações possíveis. Para cada configuração, 10 instâncias foram geradas. Portanto, um total de 1440 instâncias foram testadas.

3.2 Métrica para avaliação dos algoritmos

A métrica comumente utilizada para avaliar a qualidade das soluções obtidas por algoritmos heurísticos é a medida do Desvio Percentual Relativo (RPD) (Vallada *et al.*, 2008):

$$RPD = \frac{f_{method} - f_{best}}{f_{best}} \times 100\% \tag{2}$$

onde f_{method} é o valor da função objetivo obtido por um dado algoritmo e f_{best} é a melhor solução obtida entre todos os algoritmos comparados.

A utilização da métrica RPD em problemas que consideram a minimização de atraso pode fazer com que ocorrera divisão por zero (quando o atraso é zero para a melhor solução). Então, neste trabalho foi utilizado a medida do Índice de Desvio Relativo (RDI) (Vallada *et al.*, 2008) que é computada da seguinte maneira:

$$RDI = \frac{f_{method} - f_{best}}{f_{worst} - f_{best}} \times 100\% \tag{3}$$

onde f_{worst} é a pior solução obtida entre todos os algoritmos comparados. Com esta medida, um índice entre 0 e 100 é obtido para cada método tal que uma boa solução terá um índice muito próximo de 0. Note que se a pior e a melhor solução têm o mesmo valor, todos os métodos fornecem a melhor (mesma) solução e, assim, o valor do RDI será 0 para todos os métodos.

3.3 Calibração dos Parâmetros do algoritmo ILS_Melhorado

Para o algoritmo ILS_Melhorado, cinco parâmetros foram ajustados: N, d_{max}, γ, β e FFI. Para o algoritmo ILS_Básico foram analisados quatro parâmetros: γ, β, FFI e d. O parâmetro d é um nível de perturbação fixo e que é utilizado somente no ILS_Básico.

Os parâmetros dos algoritmos foram analisados separadamente. Para o ILS_Básico, os seguintes conjuntos de valores foram testados: $\gamma \in \{0{,}1; 0{,}3; 0{,}6; 1{,}0\}$, $\beta \in \{0; 0{,}3; 0{,}6; 1{,}0\}$, $FFI \in \{0; 1\}$ e $d \in \{n/100; n/20; n/10; n/7; n/5\}$. Já para o ILS_Melhorado, foram analisados os seguintes valores: $N \in \{1; 3; 5; 10\}$, $d_{max} \in \{n/10; n/5\}$, $\gamma \in \{0{,}1; 0{,}3; 0{,}6; 1{,}0\}$, $\beta \in \{0; 0{,}3; 0{,}6; 1{,}0\}$ e $FFI \in \{0; 1\}$.

Para determinar a melhor configuração dos valores para os parâmetros de cada algoritmo, um experimento computacional foi realizado baseado na metodologia Desenho de Experimentos (DOE) (Montgomery, 2006) onde cada fator é um parâmetro controlado. O desenho fatorial completo é usado para todos os fatores de cada algoritmo. Note que ao todo foram testadas 160 e 256 configurações de parâmetros para os algoritmos ILS_Básico e ILS_Melhorado, respectivamente.

Os experimentos foram realizados utilizando uma amostra de 100 instâncias diferentes do problema. Para cada instância os algoritmos foram rodados cinco vezes utilizando cada configuração dos parâmetros, e a o valor da função objetivo é utilizado para calcular o RPD (equação (2)) em cada uma das cinco execuções. A utilização do RPD se justifica por não se verificar experimentalmente, nestas 100 instâncias, o atraso total zero em qualquer rodada. Então, esta medida é considerada como a variável de resposta nos experimentos estatísticos. Os resultados foram analisados através da Análise de Variância (ANOVA) paramétrica.

A Figura 2 mostra o gráfico de médias resultante do teste *Tukey* da Diferença Honestamente Significativa HSD (*Honestly Significant Difference*) com nível de confiança de 95% para as configurações testadas no algoritmo ILS_Melhorado. Das 256 configurações testadas, por questão de facilidade de visualização, somente as configurações de 138 a 153 são mostradas no gráfico. Nesta figura é possível ver que existe diferença significativa entre as configurações pois há diversos intervalos que não se sobrepõe, mesmo embora não haja uma única configuração que seja estatisticamente melhor que todas as outras. Por exemplo, há diferença significativa entre as configurações 139 e 140 porque os respectivos intervalos não se sobrepõe. Para o algoritmo ILS_Melhorado a configuração 142 apresentou a melhor média e corresponde aos valores $N = 1$, $d_{max} = n/5$, $\gamma = 0{,}6$, $\beta = 0{,}6$ e $FFI = 1$. Análise semelhante foi realizada para o algoritmo ILS_Básico para o qual se chegou aos valores $\gamma = 0{,}6$, $\beta = 0{,}3$, $FFI = 1$ e $d = n/5$. Como os valores encontrados para os parâmetros $d = n/5$ e $d_{max} = n/5$ são valores limites (considerando os valores testados para os respectivos parâmetros) mais uma etapa de calibração foi realizada, visto que o maior valor possível de perturbação é $n/2 - 1$. Assim, mais três níveis para os parâmetros d e d_{max} foram testados: d e $d_{max} \in \{n/4; n/3; n/2 - 1\}$ sendo que aquele que na média obteve os melhores resultados foi $d = n/3$ e $d_{max} = n/3$. Após as duas etapas de calibração os valores dos parâmetros utilizados nos experimentos finais para os algoritmos ILS_Básico e ILS_Melhorado foram ($\gamma = 0{,}6$, $\beta = 0{,}3$, $FFI = 1$ e $d = n/3$) e ($N = 1$, $d_{max} = n/3$, $\gamma = 0{,}6$, $\beta = 0{,}6$ e $FFI = 1$), respectivamente.

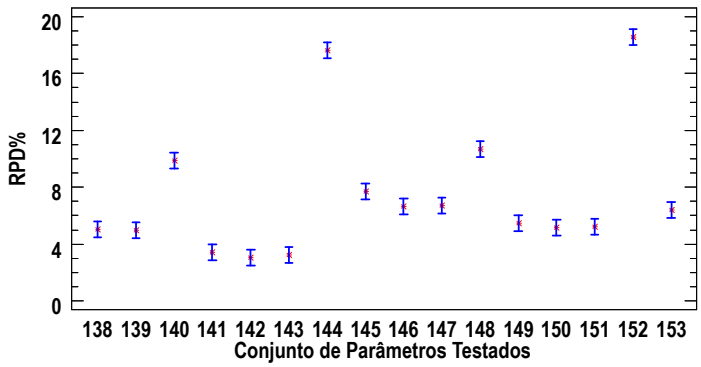

Figura 2: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 256 configurações de ILS_Melhorado. Etapa 1 da calibração.

3.4 Resultados e Comparações

Os resultados obtidos pelos algoritmos ILS_Básico, ILS_Melhorado e HGA são comparados usando o RDI (equação (3)). Todos os algoritmos foram executados com o mesmo critério de parada que é baseado em uma quantidade de tempo de CPU, que foi fixada em $500 \times n$ milissegundos. Para resolver cada instância, cada algoritmo foi executado 30 vezes. O RDI é calculado considerando o valor da função objetivo em cada repetição.

Tabela 2: Índice de Desvio Relativo (RDI) Médio.

Instância n x F	**HGA**	**ILS_Básico**	**ILS_Melhorado**
60 x 2	12,67	1,78	1,91
60 x 3	26,52	1,77	2,19
60 x 4	35,41	2,87	3,05
60 x 5	38,60	4,54	5,24
80 x 2	22,38	3,98	3,68
80 x 3	33,74	4,99	3,88
80 x 4	43,30	4,42	4,42
80 x 5	47,00	6,24	6,11
100 x 2	28,86	4,83	3,22
100 x 3	41,07	5,92	4,27
100 x 4	45,21	4,09	3,23
100 x 5	47,29	5,90	5,77
Média	35,17	4,28	3,91

A Tabela 2 mostra os resultados médios encontrados para todas as instâncias de problemas. As 1440 instâncias são agrupadas em 12 conjuntos de acordo com número de tarefas e número de famílias (veja coluna "Instância" da Tabela 2). Esta tabela mostra o quanto a média das soluções de cada algoritmo difere da melhor solução conhecida. Claramente pode-se notar que os algoritmos propostos, ILS_Básico e ILS_Melhorado, apresentam melhores resultados em comparação ao HGA para todos os problemas testados. O algoritmo ILS_Melhorado apresenta um desempenho relativamente melhor que o ILS_Básico.

Para validar os resultados obtidos pelos três algoritmos e verificar se as diferenças observadas são estatisticamente significantes, foi realizada uma Análise de Variância (ANOVA) paramétrica. O *teste-F* na tabela ANOVA irá testar se há qualquer diferença significante entre as médias do RDI (variável de resposta) encontradas pelos algoritmos (tratamentos). Foram verificadas as três pressuposições da ANOVA para que os resultados do teste sejam estatisticamente válidos: normalidade (pelo teste de *Shapiro-Wilk W*), igualdade de variância (pelo teste de *Levene*) e a independência dos resíduos (pela plotagem dos resíduos).

O resultado da ANOVA, pode ser visto na Tabela 3. Esta tabela decompõe variação total entre todas as observações, na variação devido ao efeito dos tratamentos e na variação devida ao acaso ou resíduos (veja coluna "Fonte da Variação"). O valor de F, que neste caso é igual a 41050,92, é o quociente entre o quadrado médio dos tratamentos pelo quadrado médio do resíduos. Uma vez que o valor-P do teste-F, que é o valor de interesse, é menor que 0,05, há uma diferença estatisticamente significante entre as médias do RDI de um algoritmo para outro, com um nível de significância de 95%.

A ANOVA não especifica quais algoritmos são diferentes entre si, de modo que foi realizado um teste de Comparações Múltiplas para comparar cada par de médias com um nível de confiança de 95%. A Tabela 4 mostra o resultado deste teste. A coluna "Diferença" mostra a média das amostras do primeiro algoritmo menos as do segundo. A coluna "+/- Limite" corresponde ao intervalo de incerteza para a diferença. Para qualquer

Tabela 3: Tabela ANOVA para comparação dos algoritmos HGA, ILS_Básico, ILS_Melhorado.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	19314,3	2	9657,16	**41050,92**	**0,0000**
Resíduos	20,4666	87	0,235248		
Total	19334,8	89			

par de algoritmos para no qual o valor absoluto da diferença exceda o limite significa que os algoritmos são significativamente diferentes no nível de confiança selecionado de 95%. Na tabela, a existência de diferença é indicado por um (*) na coluna "Significante". Pode-se ver que há diferença significativa entre todos os pares de médias.

Tabela 4: Teste de comparações múltiplas para o RDI.

Par de algoritmos	**Significante**	**Diferença**	**+/-Limite**
HGA - ILS_Básico	(*)	30,8922	0,298623
HGA - ILS_Melhorado	(*)	31,2565	0,298623
ILS_Básico - ILS_Melhorado	(*)	0,364265	0,298623

A mesma análise pode ser mostrada na Figura 3. Esta figura mostra o gráfico de médias resultantes do teste *Tukey* HSD. Uma vez que o intervalo para o algoritmo HGA não sobrepõe outros intervalos, a média do HGA é significativamente diferente das médias dos outros dois algoritmos. Isto é, as heurísticas propostas mostram desempenhos superiores ao método HGA.

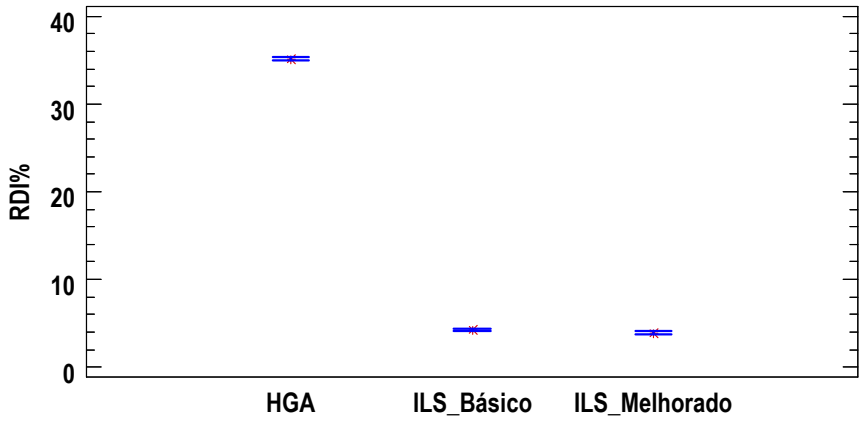

Figura 3: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos três algoritmos.

Para analisar a eficiência do controle automático da perturbação no algoritmo ILS, foi realizada uma comparação somente entre ILS_Melhorado e ILS_Básico (o RDI é recalculado conforme (3) considerando somente estes dois algoritmos). Foi realizado um *teste-t* para comparação das médias dos dois algoritmos. O nível de confiança utilizado foi de 95.0%. O *valor-P* computado foi 1,12033E-11 $< 0{,}05$ mostrando que há uma diferença significativa entre os dois algoritmos. O teste *Tukey* HSD com nível de confiança de 95% mostrado na Figura 4 também reforça o resultado. Note que a variante ILS_Melhorado tem desempenho melhor que o ILS_Básico, uma vez que não há sobreposição dos intervalos.

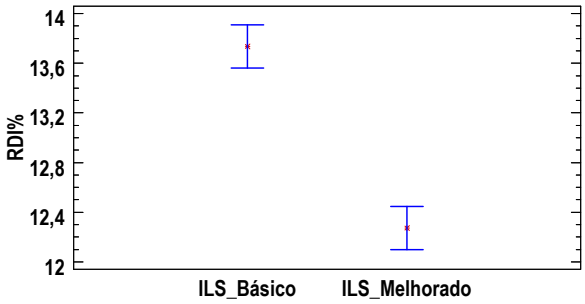

Figura 4: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos algoritmos ILS_Básico e ILS_Melhorado.

4 Conclusões

A proposta deste estudo foi examinar o problema de programação de tarefas em uma máquina com tempo de preparação dependente da sequência das famílias tal que o atraso total é minimizado. O problema $1|ST_{sd,b}|\sum T_j$ é um problema complexo de otimização combinatória com ampla aplicação nas indústrias. Em virtude da natureza NP-Difícil deste tipo de problema, um algoritmo baseado na metaheurística *Iterated Local Search* (ILS), que mantém tanto simplicidade quando generalidade, é proposto. Também foi testada uma estratégia de controle automático da força de perturbação. Os parâmetros dos algoritmos foram analisados e determinados através da metodologia Desenho de Experimentos. Para avaliar a aplicabilidade das heurísticas ILS_Básico e ILS_Melhorado propostas, seus desempenhos foram comparados com o melhor algoritmo (algoritmo genético hibrido HGA) disponível na literatura em um grande conjunto de problemas testes. Os resultados computacionais claramente indicam que a inclusão do controle automático da perturbação no ILS produz um melhoramento significativo no algoritmo. Os resultados obtidos também mostram que as heurísticas propostas são mais efetivas que o algoritmo HGA. Os resultados computacionais foram validados através de análises estatísticas, mostrando que a variante ILS_Melhorado é o melhor algoritmo. Como trabalhos futuros, devem ser feitas análises quanto ao tempo de execução do algoritmos bem como de outras distribuições na geração das instâncias e o comparativo com um modelo de programação inteira mista.

Agradecimentos

Os autores agradecem o financiamento do CNPq, FAPEMIG e CAPES.

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. *European Journal of Operational Research*, p. 985–1032.

Baker, K. R. e Magazine, M. J. (2000), Minimizing maximum lateness with job families. *European Journal of Operational Research*, v. 127, n. 1, p. 126 – 139.

Chantaravarapan, S., Gupta, J. N. D. e Smith, M. L. (2003), A hybrid genetic algorithm for minimizing total tardiness on a single machine with family setups. *POMS meeting*.

Dong, X., Huang, H. e Chen, P. (2009), An iterated local search algorithm for the permutation flow-shop problem with total flowtime criterion. *Computers and Operations Research*, v. 36, p. 1664–1669.

Du, J. e Leung, J. Y. (1990), Minimizing total tardiness on one machine is np-hard. *Math. Oper. Res.*, v. 15, n. 3, p. 483–495.

Gendreau, M. e Potvin, J.-Y. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edição. ISBN 1441916636, 9781441916631, 2010.

Graham, R., Lawler, E., Lenstra, J. e Kan, A. Optimization and approximation in deterministic sequencing and scheduling: a survey. volume 5 of *Annals of Discrete Mathematics*, p. 287 – 326. Elsevier, 1979.

Jin, F., Gupta, J. N. D., Song, S. e Wu, C. (2010), Single machine scheduling with sequence-dependent family setups to minimize maximum lateness. *JORS*, v. 61, n. 7, p. 1181–1189.

Kacem, D. Lower bounds for tardiness minimization on a single machine with family setup times. *Computational Engineering in Systems Applications, IMACS Multiconference on*, volume 1, p. 1034 –1039, 2006.

Karabati, S. e Akkan, C. (2005), Minimizing sum of completion times on a single machine with sequence-dependent family setup times. *Journal of the Operational Research Society*, v. 57, n. 3, p. 271–280.

Lourenço, H. R. (1995), Job-shop scheduling: computational study of local search and large-step optimization methods. *European Journal of Operational Research*, v. 83, p. 347–364.

Lourenço, H. R., C., M. O. e Stutzle, T. *Iterated local search*. F. Glover Eds. Kluwer Academic, 2003.

Martin, O. e Otto, S. W. (1996), Combining simulated annealing with local search heuristics. *Annals of Operations Research*, v. 63, p. 57–75.

Montgomery, D. C. *Design and Analysis of Experiments*. John Wiley & Sons. ISBN 0470088109, 2006.

Nawaz, M., Jr, E. E. E. e Ham, I. (1983), A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, v. 11, n. 1, p. 91 – 95.

Pan, J. C.-H. e Su, C.-S. (1997), Single machine scheduling with due dates and class setups. *Journal of the Chinese Institute of Engineers*, v. 20, n. 5, p. 561–572.

Rego, M. F., Souza, M. J. F. e Arroyo, J. E. C. (2012), Algoritmos multiobjetivos para o problema de sequenciamento de famÃlias de tarefas em uma máquina. *La Conferencia Latinoamericana en Informática (CLEI), Medellín Colombia*.

Ruiz, R. e Stutzle, 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*, v. 187, n. 3, p. 1143–1159.

Schaller, J. (2007), Scheduling on a single machine with family setups to minimize total tardiness. *International Journal of Production Economics*, v. 105, n. 2, p. 329–344.

Schaller, J. E. e Gupta, J. N. (2008), Single machine scheduling with family setups to minimize total earliness and tardiness. *European Journal of Operational Research*, v. 187, n. 3, p. 1050 – 1068.

Tamannaei, M. e Rasti-Barzoki, M. (2019), Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem. *Computers & Industrial Engineering*, v. 127, p. 643–656.

Vallada, E., Ruiz, R. e Minella, G. (2008), Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Comput. Oper. Res.*, v. 35, n. 4, p. 1350–1373.

Van der Veen, J. A., Woeginger, G. J. e Zhang, S. (1998), Sequencing jobs that require common resources on a single machine: a solvable case of the tsp. *Mathematical programming*, v. 82, n. 1-2, p. 235–254.