Heurística Iterated Greedy para o Problema de Sequenciamento de Lotes de Tarefas em Máquinas Paralelas

Michele Bernardino Fidelis, José Elias Claúdio Arroyo
Departamento de Informática, Universidade Federal de Viçosa(UFV)
Viçosa-Minas Gerais-MG-Brasil-36570-000
michele.bernardino@ufv.br,jarroyo@dpi.ufv.br

RESUMO

Este trabalho aborda o problema de sequenciamento de tarefas em máquinas paralelas idênticas de processamento em lotes. Um lote consiste de um grupo de tarefas que podem ser processadas simultaneamente na mesma máquina. O tamanho de um lote, número de tarefas no lote, não deve exceder a capacidade das máquinas. Neste problema tarefas possuem diferentes tempos de chegada e são classificadas em famílias incompatíveis. Tarefas da mesma família possuem os mesmos tempos de processamento e podem ser processadas em um mesmo lote. O problema consiste em determinar os lotes de tarefas e sequenciar-los nas máquinas de tal maneira que o atraso total ponderado das tarefas seja minimizado. Motivado pela alta complexidade do problema, neste trabalho é proposto uma heurística *Iterated Greedy* (IG) para determinar soluções de alta qualidade. De acordo com o que se conhece sobre o estado da arte do problema, esta é a primeira aplicação da meta-heurística IG para o problema em estudo. Experimentos computacionais são realizados a fim de comparar o desempenho da heurística IG com duas heurísticas da literatura: algoritmo memético e VNS.

PALAVRAS CHAVE. Máquinas paralelas de processamento de lotes, Iterated Greedy, Meta-heurísticas.

ABSTRACT

This paper address a job scheduling problem on identical parallel batch processing machines. A batch is a group of jobs that have to be processed jointly on a machine. The size of a batch, number of jobs in the batch, must not exceed the capacity of the machines. In this problem different job ready times and incompatible job families are considered. Jobs of the same family have identical processing times and can be processed in the same batch. The problem consists of forming job batches and schedule the batches on the machines in order to minimize the total weighted tardiness. Motivated by the computational complexity of the problem, in this paper we use a Iterated Greedy (IG) heuristic to find good quality solutions. To the best of our knowledge, this is the first application of IG meta-heuristic to solve the problem under study. Computational experiments are carried out in order to compare the performance of the heuristic IG against two heuristics from the literature: memetic algorithm and VNS.

KEYWORDS. parallel batch processing machines, Iterated Greedy, meta-heuristics.

1. Introdução

Este trabalho aborda o problema de sequenciamento de lotes de tarefas em máquinas paralelas idênticas. Esse problema consiste em agrupar tarefas em lotes para que sejam processadas simultaneamente por uma máquina. As tarefas possuem tempos de chegadas diferentes e elas são classificadas em famílias de acordo a suas incompatibilidades, isto é, tarefas de famílias diferentes não podem ser processadas no mesmo lote. Cada tarefa j possui uma data de entrega d_j na qual deseja-se que a tarefa esteja concluída, e uma prioridade w_j. Tarefas da mesma família possuem o mesmo tempo de processamento. O objetivo é sequenciar os lotes gerados nas máquinas tal que o atraso total ponderado (*total weighted tardiness (TWT)*) seja minimizado. O atraso de uma tarefa j é definido como $T_j = max(0, C_j - d_j)$, onde C_j é o tempo de conclusão da tarefa j. O TWT é definido como $TWT = \sum_j w_j T_j$. O problema em estudo é considerado NP-difícil, pois o problema mais simples de sequenciamento de tarefas em uma única máquina com a minimização do TWT também é NP-difícil [Pinedo, 2012].

Na literatura duas classes de problemas de loteamento são considerados. No primeiro tipo, chamado *s-batching*, as tarefas de um lote são processadas em série por uma máquina e o tempo de processamento do lote é a soma dos tempos de processamento das tarefas presentes no lote. Na segunda classe de problema, chamado *p-batching*, permite-se que as tarefas do lote sejam processadas simultaneamente (ou em paralelo) por uma máquina e o tempo de processamento do lote é dado pelo maior tempo de processamento das tarefas presentes no lote. Neste trabalho é abordado o *p-batching* no qual todas as tarefas de um lote devem pertencer à mesma família.

Modelos do tipo *p-batching* são aplicáveis em muitos ambientes industriais, tais como processamento de minerais, indústrias farmacêuticas, metalurgia, processamento químico, fabricação de eletrónicos e semicondutores, etc. [Mönch et al., 2011] [Mathirajan e Sivakumar, 2006]. Na literatura existem alguns trabalhos que consideram tarefas com tamanho diferentes. Jia et al. [2016] considera o problema de sequenciar n tarefas de diferentes tamanhos e de famílias incompatíveis em um conjunto de máquinas paralelas de processamento em lotes com o objetivo de minimizar o makespan. Um algoritmo *Ant Colony Optimization* (ACO) é proposto pelos autores apresentando bons resultados para o problema. Jia et al. [2015] consideram tarefas com tamanhos diferentes e máquinas de diferentes capacidades. Para minimizar o makespan eles propõem uma heurística construtiva baseada na regra *First-Fit-Decreasing* (FFD), assim como uma meta-heurística baseada em *Max-Min Ant System* (MMAS). Um algoritmo *Simulated Annealing* (SA) é proposto por Damodaran e Vélez-Gallego [2012] para minimizar o makespan, considerando tarefas com tempos de processamento diferentes, tamanhos diferentes e tempos de chegada diferente de zero.

Alguns trabalhos da literatura consideram tarefas com tamanho unitário. Lausch e Mönch [2016] abordam um problema com máquinas paralelas idênticas de processamento em lote e com tarefas pertencentes a famílias incompatíveis. A fim de minimizar o atraso total ponderado eles desenvolveram um Algoritmo Genético (AG) e um algoritmo ACO. Mönch et al. [2005] propõem duas versões diferentes de AG para minimizar o atraso total ponderado em máquinas paralelas idênticas de processamento de lotes com tarefas pertencentes a famílias incompatíveis. Malve e Uzsoy [2007] consideram tarefas com tempos de chegada dinâmico e pertencentes a diferentes famílias. Eles propõem um AG para minimizar o atraso máximo. Em Damodaran e Velez-Gallego [2010] é proposta uma heurística para minimizar o makespan em máquinas paralelas de processamento em lotes com tarefas que possuem diferentes tempos de chegada.

Este trabalho considera o problema abordado por Chiang et al. [2010]. Estes autores propuseram um algoritmo memético para minimizar o atraso total ponderado.

Bilyk et al. [2014] também consideram o problema de minimizar o atraso total ponderado, no entanto eles também consideram restrições de precedência entre as tarefas. Estes autores propuseram duas heurísticas baseadas em *Variable Neighborhood Search* (VNS) e GRASP, respectivamente, sendo que a heurística VNS apresentou o melhor desempenho. Neste artigo propomos uma heurística baseada na meta-heurística *Iterated Greedy* [Ruiz e Stützle, 2007]. Os resultados obtidos pela heurística proposta são comparados com os melhores resultados encontrados pelo algoritmo memético de Chiang et al. [2010] e o VNS de Bilyk et al. [2014].

Este artigo é organizado da seguinte forma. Na Seção 2 o problema é definido, e na Seção 3 é apresentada a descrição da heurística *Iterated Greedy*. Nas Seções 4 e 5 são apresentados os experimentos computacionais e as conclusões, respectivamente.

2. Definição do Problema

O problema é definido como em Chiang et al. [2010] da seguinte forma. Existe um conjunto $J = \{1, ..., n\}$ de n tarefas e um conjunto $M = \{1, ..., m\}$ de m máquinas idênticas, no qual cada tarefa $j \in J$ deve ser processada por apenas uma máquina $i \in M$. As tarefas possuem tamanhos unitários e as máquinas são de processamento em lote, ou seja, as tarefas podem ser agrupadas e processadas juntas em uma máquina. Cada tarefa pertence a uma família f. Todas as tarefas da mesma família f possuem o mesmo tempo de processamento p_f, portanto somente tarefas da mesma família podem ser processadas juntas. Toda tarefa j possui um tempo de chegada r_j, isto é, o processamento da tarefa somente pode começar após a chegada da tarefa. Um lote B_k com tarefas da família f estará disponível para ser processado no tempo $R_k = \max\{r_j | j \in B_k\}$. O tempo de processamento de um lote é igual ao tempo de processamento das tarefas pertencentes ao lote. O tamanho máximo do lote é denotado por B, portanto cada máquina pode processar no máximo B tarefas ao mesmo tempo. Cada tarefa j possui também um peso ou prioridade w_j e uma data de entrega d_j. Quando uma máquina começa a processar um lote B_k de tarefas, nenhuma interrupção é permitida, assim, não é possível adicionar ou retirar tarefas no lote até que o processamento do lote finalize.

O problema consiste em agrupar as tarefas em lotes e em seguida sequenciar-los nas máquinas a fim de minimizar o atraso ponderado total. O atraso da tarefa j é definido como $T_j = max(0, C_j - d_j)$, onde C_j é o tempo de conclusão da tarefa j. O atraso ponderado total é determinado por $TWT = \sum_j w_j T_j$.

3. Heurística Iterated Greedy Proposta

Iterated Greedy (IG) é uma meta-heurística proposta por Ruiz e Stützle [2007] baseada em um princípio de construção e descontrução de soluções. Esta meta-heurística foi utilizada com muito sucesso para resolver diferentes problemas de sequenciamento de tarefas [Ruiz e Stützle, 2008], [Ying et al., 2009], [Rodriguez et al., 2013], [Pan e Ruiz, 2014].

Neste trabalho, a fim de resolver o problema em estudo, é desenvolvida uma heurística baseada em IG, onde o tamanho da destruição é variável. O pseudocódigo da heurística IG é apresentado no Algoritmo 1. Os parâmetros de entrada do Algoritmo são *Criterio_de_parada* (tempo limite de execução) e d_{max} (o tamanho máximo da destruição). Observando o pseudocódigo, inicialmente uma solução inicial é gerada usando a heurística TWD [Bilyk et al., 2014] em seguida a solução é melhorada por um procedimento de busca local, resultando na solução S (linha 3). Na linha 4, o tamanho da destruição t (número de tarefas a serem retiradas da solução) e a melhor solução S_b são inicializados. Os passos entre as linhas 5 e 18 são executados iterativamente até que o critério de parada seja satisfeito. A cada iteração do algoritmo, a solução corrente S é submetida ao procedimento

de Destruição-Construção e a solução obtida é melhorada pela busca local. Se a solução S^* retornada pela busca local for melhor que a solução corrente S, o tamanho da destruição t é reinicializado em 1 e $S = S^*$. Caso contrário, o tamanho da destruição t é incrementado, $t = t + 1$ (linha 13). O valor máximo que t pode assumir é limitado por d_{max} (linha 14). Na linha 16 verifica-se o critério de aceitação. Se a solução S^* é aceita pelo critério, a solução corrente S é substituída pela solução S^*. Este critério utiliza um parâmetro T denominado temperatura que é calculada por $T = \sum_{j=1}^{n} p_j/(n*m*10)$, onde p_j é o tempo de processamento da tarefa j, n é o número de tarefas e m o número de máquinas [Ruiz e Stützle, 2007].

Algoritmo 1: Iterated Greedy

```
1  inicio
2      S = AlgoritmoTWD(S);
3      S = BuscaLocal(S);
4      t = 1; S_b = S;
5      enquanto Critério_de_parada não satisfeito faça
6          S* = DestruiçãoConstrução(S, t);
7          S* = BuscaLocal(S*);
8          se TWT(S*) < TWT(S) então
9              t = 1; S = S*;
10             se TWT(S*) < TWT(S_b) então
11                 S_b = S*;
12         senão
13             t = t + 1;
14             se t > d_max então
15                 t = 1;
16             se rand(0,1) ≤ exp(−(TWT(S*) − TWT(S_b))/T) então
17                 S = S_b
18     fim
19 fin
```

A seguir são descritos os métodos utilizados no algoritmo IG: AlgoritmoTWD, DestruiçãoConstrução e BuscaLocal.

3.1. Algoritmo TWD para a Geração da Solução Inicial

Para gerar uma solução inicial viável é utilizado o algoritmo construtivo TWD (*Time Window Decomposition*) proposto por Bilyk et al. [2014]. Esta heurística utiliza duas regras de prioridade. A primeira regra, denominada *Apparent Tardiness Cost* (ATC) proposta por Vepsalainen e Morton [1987], determina a prioridade da tarefa j ser inserida em um lote. O índice de prioridade para uma tarefa j é calculado como: $I_{\text{ATC,j}}(t) = \frac{w_j}{p_j} exp \frac{(-(d_j - p_j - t + (r_j - t)^+)^+)}{k\bar{p}}$, onde k é um parâmetro escalar, t é o tempo onde a próxima máquina estará disponível e $\bar{p}$ é a média do tempo de processamento das tarefas que ainda não foram processadas. A notação x^+ é definida como $x^+ = max(x, 0)$.

A segunda regra é utilizada para escolher um lote que será sequenciado em uma máquina. A prioridade de um lote é calculada usando a regra BATC-II [Bilyk et al., 2014]. De acordo com essa regra o valor do índice de um lote b com n_b tarefas é calculado como: $I_{\text{BATC-II},b}(t) = \frac{n_b}{B} \sum_{j \in J(b)} exp \frac{(-(d_j - p_j - t + (r_b - t)^+)^+)}{k\bar{p}}$, onde $J(b)$ é o conjunto de tarefas presentes no lote b e r_j é o maior tempo de chegada das tarefas desse lote.

A descrição dos principais passos do AlgoritmoTWD é apresentada no Algoritmo 2.

Algoritmo 2: AlgoritmoTWD

1 Quando um máquina tornar-se disponível no tempo t considere a janela de tempo $(t, t + \Delta t)$ e defina o conjunto $M(f, t, \Delta t) = \left\{j | r_j \leq t + \Delta t, f(j) = f\right\}$, isto é, tarefas da família f com tempo de chegada menor do que $t + \Delta t$.
2 Ordene as tarefas no conjunto $M(f, t, \Delta t)$ em ordem decrescente pelo valor do índice ATC e derive o conjunto $M^*(f, t, \Delta t) = \{j | j \in M(f, t, \Delta t), pos(j) \leq thres\}$ com as primeiras *thres* tarefas de cada família. $pos(j)$ é a posição da tarefa j na sequência de tarefas que são incluídas no conjunto $M(f, t, \Delta t)$.
3 Considere todas as possíveis formações de lotes das tarefas do conjunto $M^*(f, t, \Delta t, thres)$. Atribuir a cada lote um valor de índice usando BATC-II. O lote com o maior valor de índice entre as diferentes famílias é escolhido e inserido na máquina disponível. Defina uma nova janela de tempo e volte ao Passo 1 se houver tarefas que não foram processadas.

O algoritmo TWD, para avaliar todas as formações de lotes, depende dos valores dos parâmentros k, Δt e *thres*. Assim como em Bilyk et al. [2014] os valores dos parâmetros utilizados foram $k = 0.1 * h$ $(h = 1, ..., 50)$, $\Delta t = \bar{p}/2$ e $thres = 15$.

3.2. Busca Local

A etapa da busca local (BL) consiste em melhorar a solução inicial ou a solução retornada pelo método DestruiçãoConstrução. A BL utiliza 3 diferentes estruturas de vizinhanças que são apresentadas a seguir:
JobInsert: Uma tarefa é removida de um lote de uma certa família e movido para outra lote da mesma família que ainda não esteja cheio ou um novo lote é formado no fim do sequenciamento da máquina. Uma tarefa somente é movida para os lotes que começam depois do tempo de chegada da tarefa.
JobSwap: Duas tarefas da mesma família em diferentes lotes são trocadas.
BatchSwap: Dois lotes da mesma família em diferentes máquinas são trocados.

Os passos da BL utilizada pelo IG são apresentados no Algoritmo 3. O algoritmo recebe uma solução inicial S. Entre as linhas 2 e 11 o algoritmo seleciona o melhor vizinho S_{melhor} obtidas pelas três estruturas de vizinhança. A estratégia utilizada é o *best improvement*, isto é, todas as possibilidades de movimento são testados e o movimento que gerar o melhor vizinho é retornado. Se a solução S_{melhor} for melhor que a solução corrente S (ou seja, o valor da função objetivo TWT for menor), atualiza-se S e o procedimento e repetido, caso contrário, o melhor vizinho encontrado é retornado e o procedimento de BL é finalizado.

3.3. Método de Destruição-Construção

A fase de Destruição-Construção é baseada na retirada de um número de tarefas da solução corrente seguida pela reinserção gulosa (*greedy*) obtendo uma nova solução. A etapa de Destruição remove aleatoriamente t diferentes tarefas da solução corrente. Resumidamente, um lote b é selecionado de forma aleatória e então uma tarefa i presente neste lote é selecionada aleatoriamente e removida para ser inserida em um conjunto JR. Este processo é repetido t vezes obtendo uma solução parcial S_p.

A próxima etapa é a Construção, que consiste em selecionar aleatoriamente uma tarefa i do conjunto JR e procurar o *melhor* lote da mesma família onde pode ser inserida, isto é um lote que possua capacidade suficiente para receber a tarefa. O melhor lote é aquele que ao inserir a tarefa i resulte no menor valor de atraso ponderado total para a solução parcial S_p. Esse processo é repetido até que o conjunto JR fique vazio, ou seja, até que todas as tarefas do conjunto JR tenham sido reinseridas em S_p, obtendo uma solução completa.

Algoritmo 3: BuscaLocal(S)

```
1  inicio
2      enquanto melhorou = true faça
3          S_1 = JobInsert(S);
4          S_2 = JobSwap(S);
5          S_3 = BatchSwap(S);
6          S_melhor = Melhor(S_1, S_2, S_3);
7          se TWT(S_melhor) < TWT(S) então
8              S = S_melhor
9          senão
10             melhorou = false;
11     fim
12     retorna S;
13 fin
```

4. Experimentos Computacionais

Nesta seção, os resultados obtidos pelo IG são comparados com os resultados do algoritmo memético proposta por Chiang et al. [2010] e a heurística VNS proposta por Bilyk et al. [2014]. O algoritmo foi implementado em C++, compilado com g++ e os testes foram realizados em um computador Intel Core i7, CPU (4GHz), com 32 GB de RAM, sistema operacional Ubuntu 14.04, 64 bits. Como em Bilyk et al. [2014] o critério de parada utilizado para o IG é de 3 segundos. O IG foi executado 10 vezes para cada instância e os resultados médios e os melhores resultados obtidos são considerados nas comparações.

4.1. Instâncias do Problema

Os experimentos computacionais são realizados utilizando um total de 4860 instâncias produzidas e disponibilizadas por Chiang et al. [2010]. As características das instâncias são apresentadas na Tabela 1. Os resultados são analisados de acordo com o número de tarefas, número de máquinas, número de famílias, tamanho dos lotes e variação dos tempos de chegada. Para isto, as seguintes notações são utilizadas: (F1, F2, F3), (M1, M2, M3), (B1, B2) e (R1, R2, R3) denotam os números de famílias, números de máquinas, tamanhos dos lotes e tipos de tempos de chegada, respectivamente.

Tabela 1: Características das instâncias do problema

Parâmetros	***Valores***
Número de famílias (f)	3, 6, 12 denotados por (F1, F2, F3)
Número de máquinas (m)	3, 4, 5 denotados por (M1, M2, M3)
Número de tarefas (n)	180, 240, 300
Tamanho máximo do lote (B)	4, 8 denotados por (B1, B2)
Tempos de processamento por família (p_j)	2, 4, 10, 16 e 20 com probabilidade 0.2, 0.2, 0.3, 0.2 e 0.1, respectivamente
Peso por tarefa (w_j)	U(0,1)
Tempos de chegada (r_j)	$r_{ij} \sim$ U(0, $\alpha \sum p_j/(m * B)$) $\alpha = 0.25, 0.5, 0.75$ denotados por (R1, R2, R3)
Data de entrega (d_j)	$d_{ij} - r_{ij} \sim$ U(0, $\beta \sum p_j/(m * B)$) $\beta = 0.25, 0.5, 0.75$

4.2. Parâmetros do algoritmo IG

Esta seção apresenta a calibração do parâmetro d_{max} (tamanho máximo da destruição) do algoritmo IG. Este parâmetro determina o número máximo de tarefas a serem retiradas na etapa de Desconstrução-Construção. Os valores testados para d_{max} foram: $n/5, n/6, n/7, n/8$ e $n/9$, onde n é o número de tarefas. A métrica utilizada para avaliar os resultados dos experimentos é o Desvio Percentual Relativo (denotado por RPD) que é calculado como: $RPD\% = \frac{f - f_{\text{best}}}{f_{\text{best}}} \times 100\%$, onde f corresponde ao valor da função objetivo obtido pela heurística IG e f_{best} corresponde à melhor solução encontrada na execução da heurística com todos os valores do parâmetro. Para analisar os resultados do IG com os diferentes valores do parâmetro d_{max}, foi utilizada um teste de Análise de Variância paramétrica [Montgomery, 2008]. A Figura 1 apresenta o gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para a comparação da heurística IG com os diferentes valores para o parâmetro d_{max}. Observa-se que não há diferenças significativas nos resultados obtidos, no entanto é possível observar que para $d_{max} = n/7$ obtém-se a menor média. Portanto, nos próximos experimentos o IG será executado com $d_{max} = n/7$.

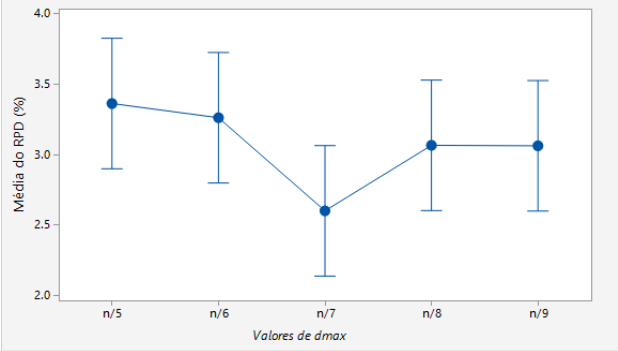


Figura 1: Gráfico de médias e intervalos HSD de *Tukey* obtidos na calibração do parâmetro d_{max}

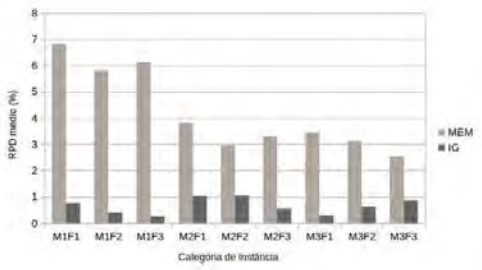
4.3. Resultados

Neste seção os resultados obtidos pelo IG, inicialmente, são comparados com os melhores resultados apresentados e disponibilizados pelo algoritmo memético (MEM) [Chiang et al., 2010]. É importante destacar que os autores do algoritmo MEM, para cada instância do problema, executaram 10 vezes o algoritmo. Os melhores resultados de todas as execuções foram disponibilizados. Para cada instância, em média o tempo de execução do MEM foi 3 segundos.

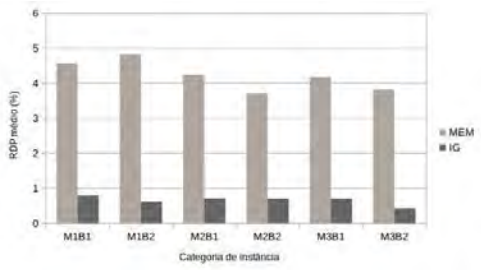
Primeiro, as melhores soluções obtidas pelo IG são comparadas com as melhores do MEM. O desempenho dos algoritmos é avaliado utilizando a medida $RPD\% = \frac{f - f_{\text{best}}}{f_{\text{best}}} \times 100\%$, onde f corresponde ao valor da solução obtida pelo algoritmo e f_{best} (solução de referência) corresponde à melhor solução encontrada entres os dois algoritmos.

Os resultados são agrupados pelo número de tarefas (n). Para cada n, definem-se 24 categorias de instâncias combinando os números de máquinas (M1, M2, M3) com os números de famílias (F1, F2, F3), tamanhos dos lotes (B1, B2) e tipos dos tempos de chegada das tarefas (R1, R2, R3). Por exemplo, a categoria M1R3 é formada por instâncias com um número pequeno de máquinas e tempos grandes de chegada. As Figuras 2, 3 e 4 apresentam as comparações do IG com o MEM para as instâncias com 180, 240 e 300 tarefas, respectivamente. Estes gráficos apresentam os RDPs médios calculados para cada categoria de instâncias. Pode-se observar que o IG apresenta o melhor desempenho para todas as categorias de instâncias. Pode ser observado que o IG apresenta piores

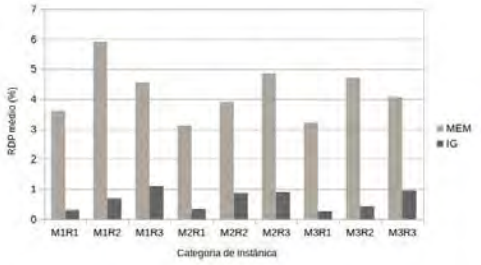
desempenhos em instâncias com tempos grades de chegada das tarefas (i.e. R3). Os melhores desempenhos do MEM são em instâncias com números de máquinas M2 e M3. Este resultado do MEM é notório nas categorias de instâncias MiFj.



(a) Categoria MiFj

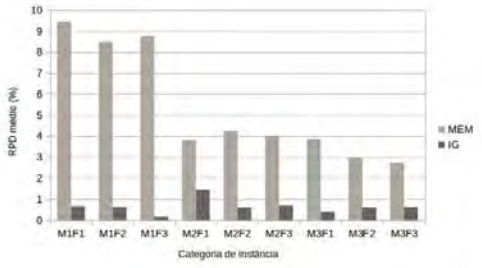


(b) Categoria MiBj

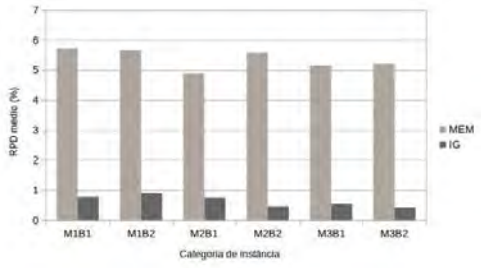


(c) Categoria MiRj

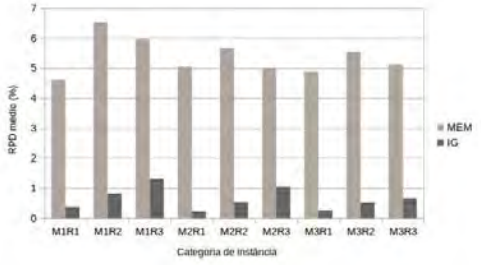
Figura 2: Comparação da qualidade da solução entre IG e MEM em instâncias com n = 180



(a) Categoria MiFj



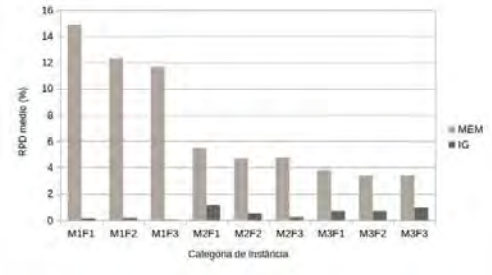
(b) Categoria MiBj



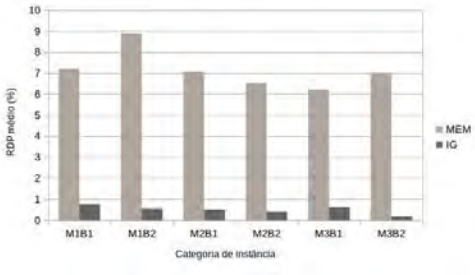
(c) Categoria MiRj

Figura 3: Comparação da qualidade da solução entre IG e MEM em instâncias com n = 240

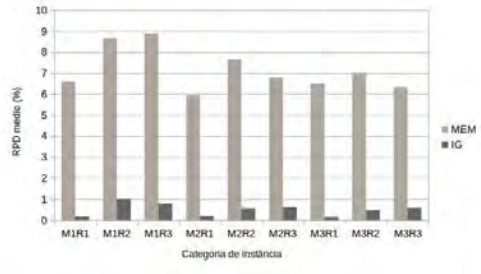
As soluções médias do IG são comparadas com as melhores soluções do algoritmo MEM. O RPD é obtido da seguinte maneira: $RPD\% = \frac{f_{\text{media}} - f_{\text{best}}}{f_{\text{best}}} \times 100\%$, onde f_{media} é a média dos valores da função objetivo obtidos nas 10 execuções do algoritmo IG e f_{best} é a



(a) Categoria MiFj



(b) Categoria MiBj



(c) Categoria MiRj

Figura 4: Comparação da qualidade da solução entre IG e MEM em instâncias com n = 300

melhor solução conhecida. Vale lembrar que nestas comparações, o RPD do MEM é obtido utilizando a melhor solução das 10 execuções do MEM.

A Tabela 2 mostra os valores dos RDPs médios dos algoritmos para cada categoria de instância. As menores médias estão em negrito. Observa-se que o IG obteve as melhores médias totais para instâncias com $n = 240$ e $n = 300$. Os resultados da Tabela 2 podem ser melhor analisadas na Figura 5. Esta Figura ilustra as médias e os intervalos HSD de Tukey com nível de confiança de 95% obtidas na execução de um teste de Análise de Variância. Pode-se notar que os resultados dos algoritmos são estatisticamente equivalentes para instâncias com 180 e 240 tarefas. No entanto, o MEM e o IG apresentaram as menores médias do RPD para as instâncias com 180 e 240 tarefas, respectivamente. Para as instâncias com 300 tarefas, os resultados médios do IG são significantemente melhores do que os resultados do MEM, pois os intervalos de confiança não se sobrepõem. Também pode ser observado que, as médias do RPD para os dois algoritmos aumentam a medida que o número de tarefas do problema é incrementado.

Em Bilyk et al. [2014], os melhores resultados de um algoritmo VNS são comparados com os resultados do MEM. Esses melhores resultados foram obtidos a partir de 10 execuções do VNS. O VNS também utiliza uma condição de parada igual a 3 segundos para cada instância. Na Tabela 3 são apresentados o percentual de melhoria do VNS (melhores resultados) e do IG (melhores resultados e resultados médios) com relação ao MEM. Esse percentual de melhoria é obtido da seguinte maneira: $\Delta(H) = \frac{f_{MEM} - f_H}{f_{MEM}} \times 100\%$, onde f_H é o valor da solução obtida pelo VNS ou IG, e f_{MEM} é o melhor resultado do algoritmo MEM. Nessa Tabela os resultados são classificados segundo os principais parâmetros do problema. As maiores médias de melhoria estão em negrito. Podemos observar que o IG (considerando os melhores resultados) apresenta uma melhoria média de 4,91 % em relação ao MEM, enquanto o VNS apresentou uma melhoria de 4,82 %. Considerando os resultados médios, o IG obteve uma melhoria média de 2,69 % em relação ao melhor resultado do MEM. Pode-se observar que o IG (melhor) e o VNS apresentaram maiores melhorias em 8 e 6 grupos de instâncias, respectivamente. Os algoritmos IG e VNS apresentaram as maiores

Tabela 2: Comparação das soluções médias do IG com as melhores soluções do MEM

Categoria de Instâncias	n = 180		n = 240		n = 300	
	MEM	IG Média	MEM	IG Média	MEM	IG Média
M1F1	6,82	**4,23**	9,44	**4,39**	14,88	**6,25**
M1F2	5,82	**3,63**	8,48	**4,82**	12,33	**4,88**
M1F3	6,14	**3,74**	8,78	**4,47**	11,67	**4,54**
M2F1	**3,81**	6,18	**3,80**	6,27	**5,48**	6,39
M2F2	**2,97**	5,51	**4,24**	5,24	**4,68**	4,99
M2F3	**3,30**	4,23	**4,02**	5,01	4,76	**4,44**
M3F1	**3,45**	5,14	**3,86**	4,82	**3,78**	5,76
M3F2	**3,11**	5,15	**2,98**	6,08	**3,38**	5,76
M3F3	**2,55**	4,88	**2,74**	5,48	**3,40**	4,96
M1B1	**4,56**	6,44	**5,72**	6,30	**7,21**	7,22
M1B2	4,83	**3,93**	5,67	**4,02**	8,89	**5,05**
M1B1	**4,24**	5,82	**4,89**	6,80	7,06	**5,84**
M2B2	3,70	3,70	5,58	**3,96**	6,53	**4,07**
M2B1	**4,18**	5,14	**5,15**	5,81	6,21	**5,37**
M2B2	**3,81**	3,42	5,21	**4,16**	7,01	**3,92**
M1R1	3,61	**2,66**	4,60	**2,62**	6,59	**2,51**
M1R2	5,91	**5,78**	6,53	**5,46**	8,67	**6,82**
M1R3	**4,56**	7,11	**5,96**	7,40	**8,89**	9,07
M2R1	3,13	**2,70**	5,04	**2,98**	5,95	**2,58**
M2R2	**3,91**	5,40	**5,68**	6,27	7,65	**5,49**
M2R3	**4,86**	6,18	**4,98**	6,89	6,79	6,79
M3R1	3,22	**2,47**	4,89	**2,70**	6,50	**2,53**
M3R2	4,70	**4,36**	**5,54**	5,80	7,00	**5,27**
M3R3	**4,06**	6,02	**5,12**	6,46	6,33	**6,18**
Médias	**4,21**	4,74	5,37	**5,18**	7,15	**5,24**

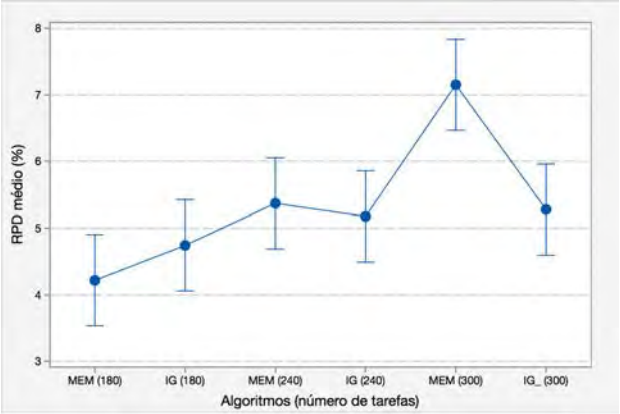


Figura 5: Gráfico de médias e intervalos HSD de *Tukey* na comparação do IG Média com o MEM

melhorias nas instâncias com 3 famílias e em instâncias com alto valor de datas de entrega. As menores melhorias foram nas instâncias com datas de entrega muito baixas (próximo de zero). Estes resultados mostram que o IG e o VNS são algoritmos competitivos.

5. Conclusões

Neste trabalho foi proposto uma heurística Iterated Greedy (IG) para a minimização do atraso total ponderado no problema de sequenciamento de lotes de tarefas em máquinas paralelas, no qual as tarefas possuem tempos diferentes de chegada e pertencem a famílias incompatíveis (tarefas de diferentes famílias não podem ser processadas juntas). A

Tabela 3: Percentual de melhoria do VNS e do IG com relação ao MEM

Fatores	Nível	VNS Melhor	IG Melhor	IG Média
Número de famílias	3	7,50	**7,95**	5,12
	6	3,68	**3,77**	1,71
	12	**3,29**	3,01	1,23
Número de máquinas	3	4,16	**5,28**	3,04
	4	**5,11**	4,73	2,56
	5	**5,19**	4,72	2,46
Tamanho máximo do lote	4	3,92	**4,85**	2,35
	8	**5,72**	4,97	3,02
Tempo de chegada (Ready time)	Próximo a zero	**5,13**	4,37	2,89
	Moderado	5,18	**5,31**	2,85
	Alto	4,16	**5,05**	2,31
Data de entrega (Due dates)	Próximo a zero	**2,90**	2,85	1,50
	Moderado	4,95	**5,15**	2,73
	Alto	6,61	**6,73**	3,83
Médias		4,82	**4,91**	2,69

heurística IG apresentada é bastante simples. O tamanho da destruição t utilizada pela heurística é variável. No início é fixado em $t = 1$, e é incrementado em 1 ($t = t + 1$) se a solução não é melhorada até $t = d_{max}$. Se a solução é melhorada em algum tamanho de destruição, t é novamente inicializado em 1. Esta ideia de variar o tamanho da destruição no IG pode ser similar à heurística VNS que utiliza várias vizinhanças durante a busca.

O desempenho do IG foi avaliado utilizando 4860 instâncias disponíveis na literatura. Os resultados obtidos foram analisados considerando os diferentes parâmetros do problema e foram comparados com os melhores resultados da literatura. A abordagem proposta foi capaz de produzir soluções de alta qualidade, obtendo uma melhoria significativa (de 4,9% em média) em relação ao algoritmo memético de [Chiang et al., 2010]. Como trabalhos futuros, pretende-se fazer uma análise de convergência do algoritmo IG considerando diferentes tempos de execução e utilizar novas estratégias de busca local.

Agradecimentos
Os autores agradecem à CAPES, FAPEMIG e ao CNPq pelo apoio ao desenvolvimento deste trabalho.

Referências

Bilyk, A., Mönch, L., e Almeder, C. (2014). Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Computers & Industrial Engineering*, 78:175–185.

Chiang, T.-C., Cheng, H.-C., e Fu, L.-C. (2010). A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival. *Computers & Operations Research*, 37(12):2257–2269.

Damodaran, P. e Velez-Gallego, M. C. (2010). Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. *The International Journal of Advanced Manufacturing Technology*, 49(9-12):1119–1128.

Damodaran, P. e Vélez-Gallego, M. C. (2012). A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert systems with Applications*, 39(1):1451–1458.

Jia, Z.-h., Li, K., e Leung, J. Y.-T. (2015). Effective heuristic for makespan minimization in parallel batch machines with non-identical capacities. *International Journal of Production Economics*, 169:1–10.

Jia, Z.-h., Wang, C., e Leung, J. Y.-T. (2016). An aco algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families. *Applied Soft Computing*, 38:395–404.

Lausch, S. e Mönch, L. (2016). Metaheuristic approaches for scheduling jobs on parallel batch processing machines. In *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, p. 187–207. Springer.

Malve, S. e Uzsoy, R. (2007). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers & Operations Research*, 34(10):3016–3028.

Mathirajan, M. e Sivakumar, A. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9-10):990–1001.

Mönch, L., Balasubramanian, H., Fowler, J. W., e Pfund, M. E. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research*, 32(11):2731–2750.

Mönch, L., Fowler, J. W., Dauzère-Pérès, S., Mason, S. J., e Rose, O. (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6):583–599.

Montgomery, D. C. (2008). *Design and analysis of experiments*. John Wiley & Sons.

Pan, Q.-K. e Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega*, 44:41–50.

Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

Rodriguez, F. J., Lozano, M., Blum, C., e GarcíA-MartíNez, C. (2013). An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers & Operations Research*, 40(7):1829–1841.

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

Ruiz, R. 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(3):1143–1159.

Vepsalainen, A. P. e Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management science*, 33(8):1035–1047.

Ying, K.-C., Lin, S.-W., e Huang, C.-Y. (2009). Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 36(3):7087–7092.