Heurísticas 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 em 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 as tarefas possuem diferentes tempos de chegada e são classificadas em famílias incompatíveis. Tarefas da mesma família possuem o mesmo tempo 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. Para resolver este problema, duas heurísticas são propostas: *Adaptive Large Neighborhood Search* (ALNS) e *Simulated Annealing* (SA). Experimentos computacionais são realizados a fim de avaliar o desempenho das heurísticas propostas, elas são comparadas com duas heurísticas da literatura: *Memetic Algorithm* and *Variable Neighborhood Search*. Os resultados demonstram que a heurística SA apresentou um desempenho superior com relação às heurísticas da literatura.

PALAVRAS CHAVE. Máquinas paralelas de processamento de lotes, *Adaptive Large Neighborhood Search*, *Simulated Annealing*, 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. To solve the problem, we proposed two search heuristic to address the problem: Adaptive Large Neighborhood Search (ALNS) and Simulated Annealing (SA). Computational experiments are carried out in order to compare the performance of heuristic ALNS and SA against two heuristics from the literature: Memetic Algorithm and Variable Neighborhood Search. The obtained results indicate that the proposed algorithm SA has a superior performance compared to algorithms proposed in the literature.

KEYWORDS. parallel batch processing machines, *Adaptive Large Neighborhood Search*, *Simulated Annealing*, meta-heuristics.

1. Introdução

Este trabalho aborda o problema de sequenciamento de lotes de tarefas em máquinas paralelas idênticas (PSLTMPI). Este problema consiste em agrupar tarefas em lotes para que sejam processadas simultaneamente por uma máquina. O objetivo é sequenciar os lotes gerados nas máquinas tal que o atraso total ponderado (*total weighted tardiness (TWT)*) seja minimizado.

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]. Segundo [Mönch et al., 2011] e [Mathirajan e Sivakumar, 2006], problemas de sequenciamento de lotes de tarefas são aplicáveis em muitos ambientes industriais tais como processamento de minerais, indústrias de farmacêuticas, metalurgia, processamento químico, fabricação de eletrônicos e semicondutores.

Na literatura alguns trabalhos abordam o problema PSLTMPI. Em [Mönch et al., 2005] dois *Genetic Algorithm* (GA) são propostos para o problema PSLTMPI enquanto em [Chiang et al., 2010] um *Memetic Algorithm* (MEM) é desenvolvido para resolver o mesmo problema. Resultados apresentados no trabalho de [Chiang et al., 2010] demonstram que o MEM supera os GA proposto por [Mönch et al., 2005] obtendo maior robustez. Já [Bilyk et al., 2014] propõe *Variable Neighborhood Search* (VNS) para o problema PSLTMPI considerando restrições de precedência, no entanto, eles também realizam uma comparação com o trabalho de [Chiang et al., 2010]. Os resultados obtidos demonstraram que o VNS é capaz de alcançar uma melhoria de 10.28% em relação ao MEM quando o critério de parada é de um minuto por instância.

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. 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. [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. 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). Já [Herr e Goel, 2016] estuda o problema de sequenciamento de tarefas em um máquina com o objetivo de minimizar o atraso total. Cada tarefa neste problema possui um tempo de processamento, uma data de chegada e pertencem a uma família. Cada tarefa requer uma certa quantidade de recurso antes do início do seu processamento.

Este trabalho considera o problema abordado por [Chiang et al., 2010]. Sua complexidade e sua importância prática são as principais motivações deste trabalho. São apresentadas duas meta-heurísticas: *Adaptive Large Neighborhood Search* (ALNS) e *Simulated Annealing* (SA) ambas, segundo nosso conhecimento, ainda não foram utilizadas para o problema PSLTMPI. Os resultados obtidos pelas heurísticas propostas são comparados com os melhores resultados encontrados pelos algoritmos re-implementados MEM de [Chiang et al., 2010] e pelo VNS de [Bilyk et al., 2014].

Este artigo é organizado da seguinte forma. Na seção 2 o problema é definido. Na seção 3 são apresentadas a descrição das meta-heurísticas ALNS e SA. Nas seções 4 e 5 são descritos os experimentos computacionais realizados e as conclusões, respectivamente.

2. Definição do Problema

O problema é definido como em [Chiang et al., 2010]: 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 simultaneamente 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 maior tempo de processamento entre as 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 do lote até que o processamento finalize.

O problema consiste em agrupar as tarefas em lotes e em seguida sequenciar-los nas máquinas a fim de minimizar o atraso total ponderado. 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 total ponderado é determinado por TWT = $\sum_{j=1}^{n} w_j T_j$.

3. Heurísticas Propostas

Neste trabalho propomos as heurísticas ALNS e SA para o problema PSLTMPI. Utilizamos a heurística *Time Window Decomposition* (TWD) proposta por Bilyk et al. [2014] em ambas heurísticas para encontrar soluções iniciais de alta qualidade. As heurísticas são descritas nas próximas subseções.

3.1. Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search (ALNS) [Pisinger e Ropke, 2007] consiste em uma adaptação da heurística baseada em *large neighborhood search*, introduzido por [Shaw, 1997]. ALNS consiste em uma heurística de busca local na qual um número de algoritmos simples compete modificar a solução corrente analisando estatisticamente o histórico de desempenho de cada método.

Em cada iteração um algoritmo é escolhido para destruir parcialmente a solução corrente e um algoritmo é escolhido para reparar a solução corrente. Esses métodos são escolhidos entre um número w de vizinhanças possíveis. ALNS utiliza *roulette wheel* para escolher um método de destruição e reparação. As vizinhanças presentes no ALNS são pesquisadas por algoritmos rápidos e simples. Segundo [Pisinger e Ropke, 2007] esse processo de busca fornece soluções de alta qualidade.

O Algoritmo 1 apresenta o pseudocódigo do ALNS segundo [Pisinger e Ropke, 2007]. O algoritmo ALNS possui seis parâmentros de entrada: *Critério_de_Parada* (tempo de CPU), *nMax* (número máximo de iterações), ρ, $\sigma 1$, $\sigma 2$ e $\sigma 3$. Na linha 1 uma solução inicial é encontrada. Vamos denotar S_{best} a melhor solução encontrada durante a pesquisa, enquanto S é a solução corrente. As iterações do algoritmo são realizadas entre as linhas 3 a 13 até que o critério de parada seja satisfeito. Entre as linhas 5 a 12 é realizado um número *nMax* de iterações. Durante cada iteração, a solução corrente S é submetida a um método

de destruição e um de reparação que são escolhidos de acordo com sua performance em iterações passadas (linha 6). O conjunto de métodos de destruição e repação são denotados por N^- e N^+, respectivamente.

Assim como em Pisinger e Ropke [2007] um peso π_i controla quais vizinhanças serão escolhidas de acordo com a frequência que ocorreram em iterações passadas. Assim, quanto mais uma vizinhança N_i tem contribuído para a melhoria da solução, um valor de peso maior é obtido, e portanto, maior será a probabilidade da vizinhança ser escolhida. *Roulette Wheel* é utilizado para escolher o par de métodos em cada iteração do ALNS. Seja π_i a pontuação passada da vizinhança i e w o número de vizinhos, então a probabilidade para escolher a vizinhança N_j é igual a $\frac{\pi_j}{\sum_{i=1}^{w}}$. Após aplicação dos métodos de destruição e reparação uma nova solução S^* é gerada a partir da solução S (linha 6). Entre as linhas 7 a 10 ocorre a verificação do critério de aceitação, se o valor do TWT da solução(S^*) for menor que o TWT da solução corrente S, atualiza-se o valor de S.

Na linha 12 os valores dos pesos dos métodos são incrementados. As vizinhanças para os métodos de destruição e construção são selecionadas independentemente. O peso de um operador mede a performance do método em cada iteração. Portanto, em cada iteração, se a solução obtida pelas iteração for a melhor global, o peso do método é incrementado por σ_1. Se a solução obtida na iteração não havia sido encontrada ainda e melhora a melhor solução atual o peso do método é incrementando em σ_2, caso contrario, o peso é incrementado por σ_3. Onde $\sigma_1, \sigma_2, \sigma_3$ são parâmetros do ALNS. Após um número $nMax$ de iterações os valores dos pesos dos métodos são atualizados, linha 13. Os pesos são alterados segundo a seguinte fórmula: $\pi_i^{t+1} = \rho\frac{\pi_i}{a_i} + (1-\rho)\pi_i^t$ onde π_i^{t+1} é o valor do novo peso para o método i, π_i o valor do peso obtido após $nMax$ iterações, a_i é o número de vezes que o método i foi utilizado durante $nMax$ iterações e ρ é o fator de reação que controla quão rápido o algoritmo de atualização do valor dos pesos reage a mudanças de resultados. A linha 13 retorna a melhor solução encontrada.

Na literatura há poucos trabalhos que utilizam o ALNS para resolver problemas de sequenciamento. Podemos citar [Lausch e Mönch, 2016] que utiliza ALNS para o problema de sequenciamento de lotes de tarefas no qual todas as tarefas estão disponíveis no tempo igual a zero. Enquanto, [Kovacs et al., 2012] and [Pillac et al., 2013] estudam *service technician routing* e *scheduling problems and the technician routing*, respectivamente.

Algoritmo 1: ALNS(*Critério_de_Parada*, $nMax, \rho, \sigma1, \sigma2, \sigma3$)

```
1  S = Solução_Inicial(S);
2  S_best = S;
3  enquanto Critério_de_Parada não Satisfeito faça
4      Iter = 0;
5      enquanto Iter < nMax faça
6          Escolhe um método de destruição N^- e um método de reparação N^+
           Gera uma nova solução S* de S após destruição e reparação
7          se TWT(S*) < TWT(S) então
8              S = S*;
9              se TWT(S*) < TWT(S_b) então
10                 S_best = S*;
11         Iter = Iter + 1;
12         Incrementa os pesos π_j de N^- e N^+;
13     Atualiza os pesos π_j de N^- e N^+;
14 Retorna S_best
```

A heurística ALNS implementada utiliza diferentes heurística de destruição e reparação. As heurísticas recebem como dados de entrada uma solução corrente S e o número de tarefas x a serem removidas da solução. As seguintes heurísticas são listadas abaixo:

Métodos de destruição:

JobRemove(S, x): Remove aleatoriamente uma tarefa de um lote. Esse processo é repetido x vezes.

JobRemoveBetter:(S, x): Escolhe aleatoriamente um lote e remove aleatoriamente uma tarefa. Esse processo é repetido x vezes.

JobRemoveReady(S, x): Escolhe aleatoriamente um lote e remove a tarefa com maior valor de tempo de chegada. Esse processo é repetido x vezes.

BatchRemove(S, x): Escolhe um lote e remove todas as tarefas pertencentes a esse lote, a probabilidade do lote ser selecionado é igual ao atrasso do lote dividido pelo atrasso total dos lotes [Kim et al., 2002]. Esse processo é repetido x vezes.

Métodos de reparação:

JobInsert(S, x): Insere a tarefa no primeiro lote da mesma família disponível. Esse processo é repetido x vezes.

JobInsertBetter(S, x): Insere a tarefa em um lote que gere o menor valor de TWT para a solução.

JobInsertChoose(S, x): Verifica se é melhor inserir uma tarefa em um lote ou formar um novo lote com a tarefa.

3.2. Solução Inicial

Para gerar uma solução inicial viável é utilizado o algoritmo construtivo (*Time Window Decomposition*) TWD 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: $I_{\text{ATC,j}}(t) = \frac{w_{\text{j}}}{p_{\text{j}}} exp\frac{(-(d_j - p_j - t + (r_j - t)^+)^+)}{k\bar{p}}$, 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: $I_{\text{BATC-II},b}(t) = \frac{n_{\text{b}}}{B} \sum_{j \in J(b)} exp\frac{(-(d_j - p_j - t + (r_b - t)^+)^+)}{k\bar{p}}$, 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 heurística TWD é explicada no Algoritmo 2:

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 $thres = 15$.

3.3. Simulated Annealing

Simulated Annealing (SA) proposto por [Kirkpatrick et al., 1983] tem sido utilizado em muitos problemas de sequenciamento [Kim et al., 2002; Li et al., 2011; Damodaran e Vélez-Gallego, 2012]. O SA utilizado neste trabalho e baseado na heurística SA proposta em [Damodaran e Vélez-Gallego, 2012] utilizada para o problema de minimizar o makespan em um grupo de máquinas idênticas de processamento de lotes. A principal ideia do SA proposto por eles é inicializar o problema com uma solução inicial com número de lotes reduzido, e pesquisar por soluções melhores sem permitir que o algoritmo acrescente um número desnecessários de lotes. O SA utilizado neste trabalho é baseado neste princípio e possui ainda um método para retirar lotes caso ocorra melhora do valor de TWT.

Algoritmo 2: TWD(k, Δt, *thres*)

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 $M^*(f, t, \Delta t) = \{j | j \in M(f, t, \Delta t), pos(j) \leq 15\}$ com as primeiras 15 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 $M^*(f, t, \Delta t, 15)$. 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.
4 Defina uma nova janela de tempo e volte ao Passo 1 se houver tarefas que não foram processadas.

Neste trabalho proposemos um algoritmo SA em que a vizinhança de soluções são geradas por seis mecanismos de pertubação. O algoritmo utiliza múltiplos reinícios, isto é, quando a temperatura tem um valor baixo (muito próximo de zero), o algoritmo é reiniciado da melhor solução obtida. O pseudocódigo do algoritmo SA é apresentado no Algoritmo 3. O SA possui três parâmetros de entrada: *Critério_de_Parada* (tempo de CPU), T_0 (temperatura inicial), α (fator de resfriamento) e *nMax* (número máximo de iterações).

Na linha 1 uma solução inicial é determinada. Vamos denotar S_{best} a melhor solução global, S^* a melhor solução para um dado número de iterações e S é a solução corrente. As iterações do algoritmo são realizadas entre as linhas 3 a 23 até que o critério de parada seja satisfeito. Na linha 4 a temperatura T é inicializada. Durante cada iteração a solução corrente S é submetida a iterações entre as linhas 5 a 22 até que o valor da temperatura T satisfaça a condição de parada. Para um dado valor de temperatura T o algoritmo realiza um número fixo de iterações *nMax* (linha 7) nas quais a solução corrente S é submetida ao processo de pertubação gerando a solução S' (linha 9). Nas linhas 10 a 16, se a solução S' melhora a corrente solução S, a solução corrente é atualizada e a melhor solução S^* obtida até o momento é atualizada se for melhorada por S'. No SA uma solução pior é aceita com uma dada probabilidade dada por $exp^{-((TWT(S')-TWT(S))/T)}$, onde $(TWT(S') - TWT(S)$ (diferença de custo entre uma nova solução e a solução corrente) e T é a constante de temperatura que depende de um instante particular. Na linha 17 a temperatura T é decrementada por uma fator α. Na Linha 18 se o valor de TWT da solução S^* for menor que o valor da melhor solução global S_{best}. Atualiza-se S_{best} (linha 19) e aplica o método *Adiciona_NovoLote(S*)* (linha 20), este método consite em formar um novo lote com uma tarefa selecionada de forma aleatória de um lote também aleatório. O novo lote formado pela tarefa é sequenciado em uma máquina aleatória. Se não ocorre melhora, então aplica-se o método *Delete_Lote(S*)* e verifica se é possível deletar um lote. Isto é realizado analisando os lotes que não estejam completos. A retirada de um lote da solução funciona de forma gulosa, primeiramente verifica se é possível retirar as tarefas do lote e se a retirada do lote levará a um valor de TWT menor que o da solução, se isso acontecer retira-se o lote e atualiza a solução, caso contrário, continua o processo de pesquisa. Na linha 23 o SA retorna a melhor solução encontrada.

Processo de Pertubação

O Processo de Pertubação utilizado no SA é baseado na pertubação utilizada em Kim et al. [2002]. A etapa de pertubação consite em um conjunto de vizinhanças, e em

Algoritmo 3: SA($Criterio\_de\_Parada, T_0, \alpha, nMax$)

```
1  S = Soluçao_Inicial(S);
2  S* = S; S_best = S;
3  enquanto Criterio_de_Parada não satisfeito faça
4      T = T0;
5      enquanto T > 0.0001 faça
6          Iter = 0;
7          enquanto Iter < nMax faça
8              Iter = Iter + 1;
9              S'= Processo_de_Pertubação(S);
10             se TWT(S') < TWT(S) então
11                 S = S';
12                 se TWT(S') < TWT(S*) então
13                     S* = S';
14             senão
15                 se random(0,1) < exp^(-((TWT(S')-TWT(S))/T)) então
16                     S = S';
17         T = αxT ;
18         se TWT(S*) < TWT(S_best) então
19             S_best = S*;
20             S= Adiciona_NovoLote(S*);
21         senão
22             S= Delele_Lote(S*);
23 Retorna S_best;
```

cada iteração um vizinho de cada vizinhança é gerado, o vizinho que gerar o menor valor de *TWT* é escolhido. As seguintes vizinhanças são utilizadas neste trabalho:

BatchInterchange: Seleciona dois lotes da mesma família de forma aleatória e troca suas posições. O primeiro lote é selecionado baseado na taxa de atrasso Kim et al. [2002], ou seja, a probabilidade de um lote ser selecionado é igual a soma do atrasso ponderado das tarefas pertences ao lote dividido pelo atrasso total ponderado do processamento.

BatchInsert: Um lote é escolhido de forma aleatória e é inserido em uma nova posição em uma máquina de forma aleatória.

BatchMerge: Seleciona dois lotes da mesma família, mescla-os e em seguida, preenche os lotes novamente.

BatchSplit: Um lote é dividido em dois e depois são inseridos em uma máquina e em uma posição escolhido aleatoriamente.

JobInterchange: Aleatoriamente seleciona e troca duas tarefas de diferentes lotes da mesma família.

JobInsert: Seleciona aleatoriamente uma tarefa em um lote, retira essa tarefa e aleatoriamente adiciona a outro lote da mesma família ou um novo lote é formado com essa tarefa.

4. Experimentos e Resultados

Nesta seção, nós descrevemos os detalhes dos experimentos utilizados neste trabalho e também é realizada uma análise dos resultados computacionais obtidos. Nós comparamos as heurísticas ALNS e SA com os algoritmos re-implementados MEM proposto por [Chiang et al., 2010] e VNS proposto por [Bilyk et al., 2014]. Os experimentos

Tabela 1: Características das instâncias do problema

Parâmetros	*Valores*
Número de Familias (f)	3, 6, 12
Número de Máquinas (m)	3, 4, 5
Número de Tarefas (n)	180, 240, 300
Tamanho Máximo do Lote (B)	4, 8
Tempo de Processamento da Família (p_j)	2, 4, 10, 16 e 20 com probabilidade 0.2, 0.2, 0.3, 0.2 e 0.1, respectivamente
Peso da Tarefa (w_j)	~U(0,1)
Tempo de Chegada (r_j)	$r_{ij} \sim \mathrm{U}(0, \alpha \sum p_j/(m * B))$ $\alpha \in \{0.25, 0.5, 0.75\}$
Data de Entrega (d_j)	$d_{ij} - r_{ij} \sim \mathrm{U}(0, \beta \sum p_j/(m * B))$ $\beta \in \{0.25, 0.5, 0.75\}$

computacionais são realizados usando um total de 4860 instâncias produzidas e disponibilizadas por [Chiang et al., 2010]. As características das instâncias são apresentadas na Table 1.

Para que as comparações realizadas neste trabalham sejam justas nós re-implementamos as heurísticas da literatura VNS e MEM. Experimentos foram realizados para verificar a equivalência entre o MEM re-implementado e o MEM da literatura. Os resultados experimentais demonstram que o valor da média dos valores dos resultados de todas as instâncias da versão re-implementada é de 937 enquanto o MEM da literatura apresenta valor de média igual a 952,79. Além disso, testes estatíticos demonstram que há diferença significativa entre a versão re-implementada do MEM e o MEM da literatura. Estes resultados são importante, uma vez todas as heurísticas são executadas na mesma máquina, podemos afirmar que a comparação realizada neste trabalho é justa.

Todas as heurísticas foram codificadas em C++ e executadas em um Cluster onde cada nó possui 2 processadores Intel(R) Xeon(R) CPU X5650(12M Cache, 2.66 GHz, 6.40 GT/s Intel(R) QPI, 6 cores, 12 threads) e 24 GB de RAM DDR3 1333 MHz. No entanto, os experimentos deste trabalho foram executados usando 1 processador, 4 threads e 8GB de RAM.

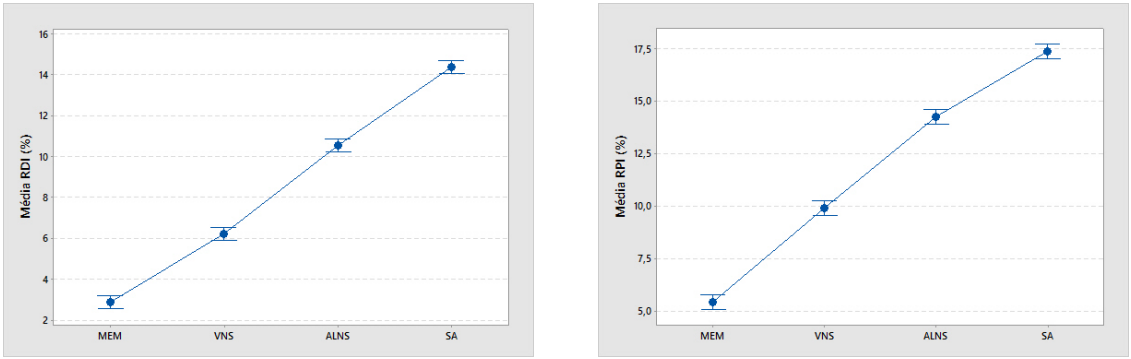
Foram realizados experimentos extensivos para determinar valores adequados para as heurísticas ALNS e SA apresentadas neste trabalho. Para esses testes o critério de parada utilizado foi igual a $0.1 * numJobs$ segundos. Todas as combinações de parâmetros de cada heurística foram testadas em um conjunto de 486 instâncias do conjunto de 4860 instâncias. Nós utilizamos o Desvio Percentual Relativo (RPD), calculado como: $RPD\% = \frac{f - f_{\text{best}}}{f_{\text{best}}} \times 100\%$, onde f corresponde ao valor da função objetivo obtido pela heurística e f_{best} corresponde a melhor solução encontrada entre as heurísticas, para determinar os melhores parâmetros a serem utilizados nas heurísticas. Após a calibração os valores utilizados na heurística ALNS α_1=33, α_2=9, α_3=13, $\rho = 0.1$ e N_w=150. No SA os seguintes parâmentros são utilizados: $\alpha = 0.5$, T_0=20000 e $IterMax = numJobs/10$.

4.1. Comparação de Performance

Os resultados obtidos pelo ALNS, SA, VNS e MEM são comparados com os melhores resultados apresentados e disponibilizados pelo algoritmo memético [Chiang et al., 2010] denotado neste trabalho por LIT e pelo VNS [Bilyk et al., 2014]. É importante destacar que os autores do algoritmo LIT, 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 enquanto o VNS Bilyk et al. [2014] utiliza o tempo de 1 minuto por instância. Neste trabalho o

critério de parada utilizado foi o tempo em função do número de tarefas. Analizamos o comportamento das heurísticas com respeito a dois critério de parada diferentes, são eles: $0.1 * numJobs$ e $0.05 * numJobs$ segundos.

O desempenho dos algoritmos é avaliado pelo percentual de melhoria *Relative Percentage Improvmente* (RPI). O RPI é calculado da seguinte forma $RPI\% = \frac{f_{\text{LIT}} - f_{\text{Heuristica}}}{f_{\text{LIT}}} \times 100\%$, onde f_{LIT} corresponde ao valor da solução obtida pelo algoritmo LIT de [Chiang et al., 2010] e $f_{Heuristica}$ (solução de referência) corresponde a melhor solução encontrada pelo algoritmo. Nós utilizamos o menor valor de *TWT* entre as dez replicações para determinar o desempenho dos algoritmos. Como em [Bilyk et al., 2014] os resultados são classificados segundo os principais parâmetros do problema.



(a) Critério_de_Parada=$0.05 * numJobs$ (b) Critério_de_Parada=$0.1 * numJobs$

Figura 1: Gráfico de Intervalos para Comparação entre as Heurísticas

A Tabela 2 mostra os valores dos RPIs médios dos algoritmos para cada categoria de instância considerando o critério de parada igual a $0.05 * numJobs$. As maiores médias estão em negrito. Observa-se que o SA obteve as melhores médias totais para instâncias com $n = 6$ e $n = 12$. Os resultados da Tabela 2 podem ser melhor analisadas na Figura 1a. 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 ALNS e SA são estatisticamente diferentes dos algoritmos VNS e MEM, pois os intervalos de confiança não se sobrepõem. O MEM e o VNS apresentaram as menores médias do RPI para as instâncias enquanto o SA aprensetou a maior média, conseguindo uma melhora de 14.41 em relação ao algoritmo LIT [Chiang et al., 2010]. Além disso, o algoritmo MEM re-implementado obteve em média melhoria de 2.98% em relação ao LIT. Enquanto o VNS re-implementado obteve em média melhoria de 6.23% em relação ao LIT.

Na Tabela 3 são apresentados os valores dos RPIs médios dos algoritmos em relação ao LIT para o critério de parada igual a $0.1 * numJobs$. Podemos observar que o SA obteve em média melhoria de 17.37% em relação ao LIT, enquanto o MEM re-implementado apresentou uma melhoria de 5,43%. Considerando os resultados médios, o VNS obteve uma melhoria média de 9.90% E O ALNS uma melhoria de 14.25% em relação ao LIT da literatura. Observando a Figura 1b podemos inferir que os resultados dos algoritmos ALNS e SA ainda permanecem estatisticamente diferentes dos algoritmos VNS e MEM re-implementados, pois os intervalos de confiança não se sobrepõem. O MEM e o VNS apresentaram as menores médias de RPI para as instâncias enquanto o SA apresentou as melhores melhorias (RPIs). Estes resultados demonstram que o ALNS e o SA superam os algoritmos VNS e MEM.

Tabela 2: Percentual de melhoria das heurísticas com relação ao MEM com critério_de_Parada = $0.05 * numJobs$

Fator	Nível	MEM	VNS	ALNS	SA
Número de Famílias	3	1.47	3.61	5.70	**9.13**
	6	5.23	10.23	16.83	**21.93**
	12	1.97	4.86	9.16	**12.18**
Número de Máquinas	3	3.12	6.56	11.39	**15.51**
	4	2.78	6.09	10.40	**14.32**
	5	2.76	6.05	9.91	**13.41**
Tamanho Máximo do Lote	4	2.38	6.15	11.65	**17.47**
	8	3.40	6.31	9.48	**11.36**
Tempo de Chegada	Próximo de zero	2.93	5.19	6.97	**10.35**
	Moderado	3.01	6.39	10.03	**14.19**
	Alto	2.73	7.12	14.69	**18.70**
Data de Entrega	Próximo de zero	1.54	3.25	4.76	**6.63**
	Moderado	2.89	5.76	8.83	**12.24**
	Alto	4.23	9.68	18.10	**24.36**
Média		2.89	6.23	10.56	**14.41**

Tabela 3: Percentual de melhoria das heurísticas com relação ao MEM com critério_de_Parada = $0.1 * numJobs$

Fator	Nível	MEM	VNS	ALNS	SA
Número de Famílias	3	3.79	6.85	9.70	**12.61**
	6	8.09	13.65	20.03	**23.66**
	12	4.40	9.21	13.01	**15.83**
Número de Máquinas	3	5.87	10.43	15.47	**18.82**
	4	5.32	9.67	13.81	**17.16**
	5	5.09	9.61	13.46	**16.12**
Tamanho Máximo do Lotes	4	5.97	10.44	16.46	**20.94**
	8	4.89	9.37	12.03	**13.80**
Tempo de Chegada	Próximo de zero	4.71	7.66	9.46	**11.81**
	Moderado	5.88	9.81	13.94	**17.48**
	Alto	5.69	12.23	19.32	**22.18**
Data de Entrega	Próximo de zero	2.97	4.91	6.97	**8.64**
	Moderado	5.43	8.71	12.63	**15.29**
	Alto	7.89	16.08	23.13	**28.18**
Média		5.43	9.90	14.25	**17.37**

5. Conclusões

Neste trabalho foram propostas as heurísticas ALNS e SA para 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).

O desempenho das heurísticas ALNS e SA foram comparados com o algoritmo memético de [Chiang et al., 2010] e VNS de [Bilyk et al., 2014]. Os desempenhos dos algoritmos foram avaliados considerando o percentual de melhoria com relação aos melhores resultados disponívies na literatura. As abordagens propostas foram capazes de produzir soluções de alta qualidade. O SA foi o algoritmo que apresentou o melhor desempenho seguido ALNS. O SA obteve uma melhoria significativa (de 17.37% em média) enquanto o ALNS obteve uma melhoria significativa (de 14.25% em média) em relação ao algoritmo memético de [Chiang et al., 2010].

Como trabalhos futuros, pretende-se fazer uma análise de convergência das heurísticas ALNS e SA e considerar restrições de precedência de tarefas.

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.

Herr, O. e Goel, A. (2016). Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints. *European Journal of Operational Research*, 248(1):123–135.

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.

Kim, D.-W., Kim, K.-H., Jang, W., e Chen, F. F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18(3):223–231.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983). Optimization by simmulated annealing. *science*, 220(4598):671–680.

Klemmt, A., Weigert, G., Almeder, C., e Mönch, L. (2009). A comparison of mip-based decomposition techniques and vns approaches for batch scheduling problems. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, p. 1686–1694. IEEE.

Kovacs, A. A., Parragh, S. N., Doerner, K. F., e Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15(5):579–600.

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.

Li, K., Yang, S.-L., e Ma, H.-W. (2011). A simulated annealing approach to minimize the maximum lateness on uniform parallel machines. *Mathematical and Computer Modelling*, 53(5):854–860.

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.

Pillac, V., Gueret, C., e Medaglia, A. L. (2013). A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535.

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

Pisinger, D. e Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*.

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