LII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2020

PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM TEMPOS DE PREPARAÇÃO, RESTRIÇÕES DE RECURSO E MINIMIZAÇÃO DO ATRASO TOTAL

Júlio C. S. N. Pinheiro[1], José E. C. Arroyo[2], Matheus F. Araujo[3]
Departamento de Informática
Universidade Federal de Viçosa
Av. Peter Henry Rolfs, s/n, 36570-900,Viçosa-MG, Brasil
julio.pinheiro@ufv.br[1], jarroyo@ufv.br[2], matheus.f.freitas@ufv.br[3]

RESUMO

Neste trabalho é abordado o problema da programação de tarefas em máquinas paralelas não-relacionadas sujeitas a tempos de preparação e restrições de recurso. Tarefas são classificadas em famílias e tempos de preparação são necessários entre o processamento de duas tarefas de famílias distintas. A execução de uma tarefa demanda uma quantidade de recurso da máquina responsável, que é suprida por processos superiores. O objetivo é encontrar uma programação de tarefas cujo atraso total seja mínimo. Uma formulação matemática de programação linear inteira-mista foi desenvolvida para o problema. Por se tratar de um problema NP-difícil, métodos exatos são ineficientes. Deste modo, desenvolve-se dois métodos resolutivos baseados nas meta-heurísticas *Simulated Annealing* e *Iterated Greedy*. Experimentos computacionais mostraram que os algoritmos apresentados resolvem satisfatoriamente o problema abordado.

PALAVRAS CHAVE. Programação de tarefas. Máquinas paralelas não-relacionadas. Atraso total. Heurísticas.

Tópicos - Meta-heurísticas. Otimização Combinatória.

ABSTRACT

This work considers the unrelated parallel machines scheduling problem with family setups and resource constraints. In this problem, jobs are grouped into families and setup times are required between jobs belonging to different families. The processing of a job requires a certain amount of resource from a machine, which is supplied by upstream processes. The total resource consumed by the jobs processed on a machine must not exceed the resource supply up. The objective is to schedule the jobs on the machines so that the total tardiness of the jobs is minimized. In attempting to obtain optimal solutions for small size instances, a mixed integer linear programming (MILP) model is proposed. Since the problem is a NP-hard combinatorial optimization problem, to obtain near optimal solutions we develop two meta-heuristic algorithms based on Simulated Annealing and Iterated Greedy. Computational experiments show that the presented algorithms can be successfully applied to the problem.

KEYWORDS. Scheduling. Unrelated parallel machines. Total tardiness. Heuristics.

Paper topics - Meta-heuristics. Combinatorial Optimization.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

1. Introdução

Neste trabalho considera-se o problema da programação de tarefas em máquinas paralelas não-relacionadas, conhecido como UPMSP (*unrelated parallel machine scheduling problem*). Nesta categoria de problemas, as máquinas funcionam simultaneamente durante o expediente de produção, porém, atuam com velocidades e características distintas umas das outras. No problema abordado, um conjunto de tarefas deve ser alocado em máquinas paralelas, independentes e não-preemptivas com o objetivo de minimizar o atraso total da programação. Cada tarefa requer da máquina responsável uma quantidade de recurso que será consumido linearmente durante todo o processamento da tarefa. Os recursos de cada máquina são supridos de forma constante durante todo o expediente de produção. Não é permitido que falte recurso durante a execução de uma tarefa, deste modo, uma tarefa poderá ter seu início adiado caso seja necessário. Em resumo, durante todo o expediente produtivo a demanda cumulativa de recursos das tarefas alocadas em uma máquina não deve exceder a oferta cumulativa da mesma. As tarefas são classificadas em famílias de configuração, de modo que o processamento consecutivo de duas tarefas de famílias distintas em uma mesma máquina deverá ser entremeado de um tempo de preparação (*setup time*). Datas de entregas são atribuídas às tarefas; o atraso de uma tarefa (*tardiness*) é definido como a diferença positiva entre seu tempo de conclusão e sua data de entrega. O atraso total da programação (*total tardiness*) é a soma dos atrasos individuais das tarefas.

O contexto prático do problema estudado surge no processo de lingotamento contínuo da produção de aço [Jiang et al., 2018]. Máquinas de lingotamento contínuo são alimentadas por panelas de aço fundido cujas ordens de serviço determinam suas datas limites de entrega. Panelas carregadas com tipos de aço semelhantes são processadas consecutivamente em uma máquina. Contudo, caso haja intercalação de panelas cujos aços possuem graduações fisico-químicas distintas, ações de preparação são necessárias à máquina operante. O aço líquido é fornecido pelo alto-forno e, em nenhum momento do processo de produção é permitido que se consuma mais aço fundido que o fornecido pelo alto-forno [Herr e Goel, 2016].

A complexidade do problema abordado pode ser avaliada da seguinte maneira: simplificando o problema para somente uma máquina com recursos virtualmente ilimitados e tarefas pertencentes a somente uma família de configuração, estaremos tratando o problema da minimização do atraso total de uma programação de tarefas em uma única máquina, que é conhecidamente NP-difícil [Du e Leung, 1990]. Portanto, segue que o problema de interesse deste trabalho também é NP-difícil.

Até onde sabemos, o problema da programação de tarefas em máquinas paralelas não-relacionadas sujeitas a tempos de preparação e restrições de recurso para minimizar o atraso total não foi previamente estudado na literatura. O caso simples de uma única máquina foi apresentado em [Herr e Goel, 2016]. Naquele trabalho, os autores apresentaram um modelo de programação linear inteira-mista (MILP) e um método heurístico baseado na meta-heurística *iterated local search* (ILS) para abordar o problema. Posteriormente, [Pinheiro e Arroyo, 2020] apresentaram a meta-heurística *iterated greedy* (IG) como alternativa mais eficaz para a obtenção de soluções para o problema.

Este trabalho busca estender o problema da programação em uma uma única máquina para um cenário verossímil de produção contendo várias máquinas paralelas e independentes. Neste artigo apresentamos uma formulação matemática de programação linear inteira-mista para resolução exata do problema, como também duas meta-heurísticas para a busca de soluções.

O restante deste artigo está organizado da seguinte maneira. Seção 2 apresenta a descrição do problema e a formulação matemática. Seção 3 apresenta as meta-heurísticas propostas

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

neste trabalho: *simulated annealing* (SA) e *iterated greedy* (IG). Seção 4 descreve a geração das instâncias-problema e a condução experimental. Seção 5 apresenta os resultados e sua discussão. Finalmente, as conclusões deste trabalho e as sugestões de estudos futuros são apresentados na Seção 6.

2. Descrição do Problema

No problema em questão, um conjunto J de n tarefas deve ser distribuído em um conjunto M de m máquinas paralelas não-relacionadas. Cada tarefa $j \in J$ possui: uma data de entrega d_j; tempos de processamento p_{jk}, $\forall k \in M$; e uma demanda q_j de recurso necessário à sua execução. Para qualquer par de tarefas $i, j \in J$ processadas consecutivamente em uma máquina k, se $f_i \neq f_j$ (famílias distintas), um tempo de preparação $s_{ijk} > 0$ é necessário entre as execuções de i e j. Se i e j pertencem à mesma família, então $s_{ijk} = 0$. Todas as tarefas $j \in J$ e todas as máquinas $k \in M$ possuem tempos de chegada $t = 0$, ou seja, estão disponíveis desde o início do expediente produtivo. O processamento das tarefas é não-preemptivo e cada tarefa deve ser alocada somente uma vez em somente uma máquina. O suprimento de recursos para uma máquina k segue a equação:

$$R_k(t) = r_k^* + t.r_k, \tag{1}$$

onde r_k^* é o recurso inicial, r_k é a taxa de incremento por unidade de tempo, e t é o tempo decorrido desde o início da produção ($t \geq 0$). Uma tarefa j consome os recursos de uma máquina k a uma taxa constante q_j/p_{jk}. Para que a restrição de recurso de uma máquina k seja satisfeita durante todo o processamento de uma tarefa j, a seguinte relação deve ser observada:

$$R_k(t) \geq Q_{ik} + t.\frac{q_j}{p_{jk}}, \quad \forall t \in [C_{jk} - p_{jk}, C_{jk}]. \tag{2}$$

onde: i é a tarefa predecessora de j; Q_{ik} é a demanda cumulativa de recurso de todas as tarefas até i (inclusivo); e C_{jk} é o tempo de conclusão de j. Por fim, o objetivo é encontrar uma programação de tarefas em cada máquina de modo a minimizar a soma dos atrasos das tarefas.

2.1. Formulação Matemática

Um modelo de programação linear inteira-mista (MILP) é desenvolvido para o problema. Para propósitos de modelagem, uma tarefa artificial é incluída no modelo. Essa tarefa artificial será predecessora e sucessora da primeira e última tarefa, respectivamente, de cada sequência de cada máquina. As variáveis de decisão são as seguintes: a variável binária x_{ijk} denota a precedência da tarefa i sobre a tarefa j na programação da máquina k (1, se sim; 0, caso contrário); Q_{jk} é a demanda cumulativa das tarefas até j (incluso) alocadas na máquina k; C_{jk} é o tempo de conclusão da tarefa j na máquina k; e T_j é o atraso positivo da tarefa j.

Portanto, temos:

$$\min \sum_{i=1}^{n} T_i \tag{3}$$

sujeito a

$$\sum_{k=1}^{m} \sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ijk} = 1 \qquad \forall j \in J \tag{4}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{n} x_{ihk} - \sum_{\substack{j=0 \\ j \neq h}}^{n} x_{hjk} = 0 \qquad \forall h \in J, \forall k \in M \tag{5}$$

$$\sum_{j=1}^{n} x_{0jk} \leq 1 \qquad \forall k \in M \tag{6}$$

$$C_{0k} = 0 \qquad \forall k \in M \tag{7}$$

$$C_{jk} \geq 0 \qquad \forall j \in J, \forall k \in M \tag{8}$$

$$C_{jk} \geq C_{ik} + s_{ijk} + p_{jk} - (1 - x_{ijk})G \qquad \forall i \in J \cup \{0\}, \forall j \in J, \forall k \in M \tag{9}$$

$$Q_{0k} = 0 \qquad \forall k \in M \tag{10}$$

$$Q_{jk} \geq 0 \qquad \forall j \in J, \forall k \in M \tag{11}$$

$$Q_{jk} \geq Q_{ik} + q_j - (1 - x_{ijk})G \qquad \forall i \in J \cup \{0\}, \forall j \in J, \forall k \in M \tag{12}$$

$$Q_{jk} \leq r_k^* + r_k C_{jk} \qquad \forall j \in J, \forall k \in M \tag{13}$$

$$T_j \geq 0 \qquad \forall j \in J \tag{14}$$

$$T_j \geq C_{jk} - d_j \qquad \forall j \in J, \forall k \in M \tag{15}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in J \cup \{0\}, \forall k \in M \tag{16}$$

A função objetivo (3) é a minimização da soma dos atrasos das tarefas. (4) determina que uma tarefa seja processada em somente uma máquina. (5) garante que uma tarefa deve ser precedida e sucedida somente uma vez; (6) determina que caso uma máquina seja utilizada, esta terá uma sequência única. (7), (8) e (9) determinam que o tempo de conclusão de uma tarefa será pelo menos o tempo de conclusão da tarefa anterior, somados o tempo de preparação (se houver) e o tempo de processamento da própria tarefa. (10), (11) e (12) garantem que para cada tarefa escalada em uma máquina, a demanda cumulativa até tal tarefa (inclusive) será pelo menos a demanda cumulativa até a tarefa anterior somada a demanda da própria tarefa. (13) determina que, durante todo o processamento de uma tarefa, a demanda cumulativa total de recursos de uma máquina não deve exceder a quantidade de recursos disponíveis na mesma. (14) e (15) indicam que o atraso de uma tarefa é a diferença positiva entre o tempo de conclusão e a data de entrega da própria tarefa. (16) garante a natureza binária de x_{ijk}.

3. Métodos heurísticos

Por se tratar de um problema NP-difícil, formulações exatas como a apresentada na Seção 2.1 possuem resoluções ineficientes, pois não retornam soluções viáveis em tempo hábil para instâncias grandes. Com o objetivo de encontrar soluções sub-ótimas para o problema em estudo propomos neste trabalho a aplicação de dois métodos resolutivos baseados em meta-heurísticas: *simulated annealing* (SA) e *iterated greedy* (IG). Esses métodos exploram iterativamente o espaço de busca por meio da aplicação de movimentos pré-definidos (vizinhanças) em soluções viáveis de modo que novas soluções são obtidas a partir das soluções encontradas anteriormente.

O ponto de partida desses métodos geralmente são soluções obtidas aleatoriamente ou heuristicamente definidas. Neste trabalho as soluções iniciais são construídas pela regra *earliest due date* (EDD), ou regra da data de entrega mais próxima. As tarefas são ordenadas ascendentemente por suas datas de entrega. Iteravamente, a tarefa com a data de entrega mais próxima é alocada à máquina que contribui com o menor incremento de atraso positivo à programação parcial construída (empates são decididos pelo menor *makespan*).

3.1. *Simulated Annealing* (SA)

SA é uma meta-heurística inspirada no processo de recozimento (*annealing*) utilizado em metalurgia para obtenção de estados de baixa energia de um sólido [Aarts et al., 2005]. A estratégia empregada neste método conta com a capacidade de evadir ótimos locais e impedir terminações precoces. Essa habilidade é alcançada pela aceitação de soluções de baixa qualidade durante a busca. A probabilidade de aceitação dessas soluções decrescem à medida que o procedimento de busca avança. O pseudocódigo é descrito no Algoritmo 1.

A heurística SA proposta conta com três parâmetros de inicialização: T (temperatura inicial); $\alpha \in [0, 1)$ (fator de arrefecimento); e β (fator de parada). Nos passos 1 e 2, a solução inicial é obtida e armazenada como soluções globais e corrente. O *loop* mais externo decrementa a temperatura inicial até que esta atinja um valor mínimo (passos 3 a 16). O espaço de soluções é explorado no *loop* mais interno (passos 5 a 15). No passo 6, uma solução vizinha é obtida randomicamente aplicando-se uma estrutura de vizinhança local N_i previamente selecionada no passo 4. Se a solução vizinha melhora a solução corrente ($\Delta < 0$), então a solução corrente é atualizada (passo 9). Caso a solução vizinha seja de pior qualidade, a solução corrente possui probabilidade $\exp(-\frac{\Delta}{T})$ de ser atualizada. Os passos 10 e 11 atualizam a solução global. A fase de melhoria termina após $\beta * n$ iterações, onde n é o número de tarefas da instância em resolução. No passo 14, a temperatura é diminuída pelo fator α e a fase de melhoria (*loop* mais interno) é reiniciada. O algoritmo termina quando a temperatura atinge uma valor mínimo $T_0 = 0.001$.

A temperatura inicial é obtida em uma fase de pré-processamento descrita na próxima subseção. As estruturas de vizinhança utilizadas estão descritas na Seção 3.3.

Algorithm 1 Simulated Annealing (T, α, β)

1: Determine a solução inicial S_{best}
2: Faça $S_1 \leftarrow S_{best}$
3: **repeat**
4: Escolha uma vizinhança $\mathcal{N}_i$
5: **for** $k = 1$ **to** βn **do**
6: Faça $S_2 \leftarrow \mathcal{N}_i(S_1)$
7: Calcule $\Delta = f(S_2) - f(S_1)$
8: **if** $\Delta < 0$ **or** $p \in [0, 1] < \exp(-\frac{\Delta}{T})$ **then**
9: Faça $S_1 \leftarrow S_2$
10: **if** $f(S_2) < f(S_{best})$ **then**
11: Faça $S_{best} \leftarrow S_2$
12: **end if**
13: **end if**
14: Faça $T \leftarrow T * \alpha$
15: **end for**
16: **until** $T \leq T_0$
17: **return** S_{best}

Algorithm 2 Temperatura inicial (β, γ, δ)

1: Seja S_1 a solução inicial
2: Faça $T \leftarrow T_{min}$
3: **loop**
4: $count \leftarrow 0$
5: **for** $k = 1$ **to** βn **do**
6: Escolha uma vizinhança $\mathcal{N}_i$
7: Faça $S_2 \leftarrow \mathcal{N}_i(S_1)$
8: Calcule $\Delta = f(S_2) - f(S_1)$
9: **if** $\Delta < 0$ **or** $p \in [0, 1] < \exp(-\frac{\Delta}{T})$ **then**
10: $count \leftarrow count + 1$
11: **end if**
12: **end for**
13: **if** $count \geq \gamma(\beta * n)$ **then**
14: break loop
15: **end if**
16: $T \leftarrow T * \delta$
17: **end loop**
18: **return** T

3.1.1. Temperatura Inicial

O processo de obtenção da temperatura inicial é descrito no Algoritmo 2. A partir de uma solução e temperatura iniciais, o procedimento contabiliza a quantidade de soluções vizinhas que

são aceitas em $\beta * n$ iterações utilizando a temperatura corrente. Caso a contagem não atinja um número mínimo $\gamma(\beta * n)$, então a temperatura corrente é aumentada por um fator δ e a contagem é reiniciada. Esse procedimento é repetido até que se atinja uma temperatura ideal em que um número mínimo de soluções vizinhas são aceitas. A temperatura final do algoritmo será a temperatura inicial do método *simulated annealing*. Os parâmetros de inicialização do Algoritmo 2 são: γ (proporção mínima de soluções que devem ser aceitas); δ (fator de aquecimento). O valor β utilizado é o mesmo de Algoritmo 1.

3.2. *Iterated Greedy* (IG)

A meta-heurística IG foi primeiramente utilizada em [Ruiz e Stützle, 2007] para resolver o problema da programação de produção *flow shop* permutacional, e desde então foi utilizada em diversos outros problemas de sequenciamento [Fanjul-Peyro e Ruiz, 2010; Arroyo et al., 2019; Pinheiro e Arroyo, 2020]. Esse procedimento consiste em sucessivos movimentos de subida de encosta (*hill climbing*) em soluções que foram destruídas e reconstruídas anteriormente. Algoritmo 3 apresenta o pseudocódigo de nossa implementação.

O procedimento inicia-se com a obtenção da solução inicial (passo 1) e, logo após, efetua uma série de iterações (passos 4 a 14) que consistem de fases de destruição (passo 5), reconstrução (passo 6) e busca local (passo 7). O método de destruição remove $\alpha \times n$ tarefas da programação corrente, onde $0 < \alpha \leq 1$ é um parâmetro de inicialização. A reconstrução é feita reinserindo gulosamente as tarefas removidas. Isso é feito da seguinte maneira: cada tarefa removida é reinserida em todas as posições possíveis, gerando soluções parciais; a solução parcial cujo atraso total seja o menor será escolhida; repete-se até que se obtenha uma solução integral. A solução reconstruída é melhorada por um procedimento de busca local descrito na Seção 3.2.1. Passos 8 a 13 atualizam a solução global e a contagem de iterações. O algoritmo IG termina quando houver um número k_{max} (parâmetro de inicialização) de iterações consecutivas sem nenhuma melhoria da solução global.

Algorithm 3 Iterated Greedy (k_{max}, α)

1: Determine a solução inicial S_{best}
2: Faça $S_1 \leftarrow S_{best}$
3: Inicialize $k \leftarrow 0$
4: **while** $k < k_{max}$ **do**
5: Remova $\alpha \times n$ tarefas de S_1
6: Reinsira as tarefas em S_1 gulosamente
7: Faça busca local em S_1
8: **if** $f(S_1) < f(S_{best})$ **then**
9: $S_{best} \leftarrow S_1$
10: $k \leftarrow 0$
11: **else**
12: $k \leftarrow k + 1$
13: **end if**
14: **end while**
15: **return** S_{best}

Algorithm 4 Busca local (S, k_{max})

1: $P_i \leftarrow 1/|V|, c_i \leftarrow 0, \quad \forall i = 1 \ldots |V|$
2: Inicialize $k \leftarrow 0$
3: **while** $k < k_{max}$ **do**
4: Escolha uma vizinhança $\mathcal{N}_i$ com probabilidade P_i
5: Faça $S_1 \leftarrow \mathcal{N}_i(S)$
6: **if** $f(S_1) < f(S)$ **then**
7: $S \leftarrow S_1, \quad k \leftarrow 0$
8: $c_i \leftarrow c_i + 1$
9: $P_i = c_i / \sum_j c_j, \quad \forall i, j = 1 \ldots |V|$
10: **else**
11: $k \leftarrow k + 1$
12: **end if**
13: **end while**
14: **return** S

3.2.1. Busca Local

Este procedimento de busca local é usado internamente pela heurística IG e está descrito no Algoritmo 4. Para a geração de novas soluções é utilizado um conjunto V de estruturas de vizinhanças descritas na Seção 3.3. A aplicação dessas vizinhanças ocorre de maneira probabilística.

Para cada vizinhança i, uma probabilidade de seleção P_i e um contador de melhorias c_i são definidos (passo 1). Todas as vizinhanças possuem, inicialmente, a mesma probabilidade de seleção. Uma vizinhança é escolhida (passo 4) e aplicada à solução corrente (passo 5). Se houve uma melhora na solução, a solução global é atualizada, o contador da vizinhança utilizada é incrementado e as probabilidades são atualizadas (passos 6 a 9). À medida que a busca local avança, estruturas que geram soluções com melhorias mais frequentemente terão probabilidade maior de serem escolhidas. A busca local termina quando se atinge k_{max} iterações consecutivas sem melhorias na solução global.

3.3. Estruturas de Vizinhança

As estruturas de vizinhanças aplicadas à programação de tarefas em máquina paralelas são as seguintes. *Job Insertion* (N_1) retira uma tarefa de uma máquina e a reinsere em outra posição em alguma máquina. *Batch Insertion* (N_2) retira e reinsere um lote de tarefas. *Job Swap* (N_3) seleciona um par de tarefas (mesma máquina ou diferentes) e as trocam de posições. *Batch Swap* (N_4) é semelhante à anterior (troca lotes). *Batch Merge* (N_5) seleciona dois lotes de tarefas de mesma família, os combinam e insere o lote resultante em alguma máquina. Finalmente, *Batch Break* (N_6) seleciona um lote de tarefas, divide-o em dois lotes menores e os insere em diferentes posições de uma mesma máquina ou de máquinas diferentes.

4. Experimentos Computacionais

Nesta seção são descritos os experimentos computacionais conduzidos para estudar a eficácia dos métodos de resolução propostos. Esses métodos foram testados em um conjunto de 864 instâncias-problema geradas neste trabalho. O modelo MILP foi resolvido pelo *solver* IBM ILOG CPLEX 12.8, distribuído em 4 *threads* e tempo limite de 3600 segundos. As heurísticas foram codificadas em C++. Todos os experimentos foram executados em máquinas Xeon(R) CPU ES-1620 @ 3.50GHz x 8, 96GB RAM com sistema operacional Ubuntu 14.04 LTS 64 bits. A qualidade das soluções foi avaliada de acordo com o *Relative Deviation Index* (RDI):

$$\text{RDI}\ (\%) = 100 \times \frac{f_{method} - f_{best}}{f_{worst} - f_{best}}, \tag{17}$$

onde f_{best} e f_{worst} são a melhor e a pior solução, respectivamente, obtidas entre todos os algoritmos comparados. Esse método de avaliação produz resultados entre 0 e 100: valores próximos de 0 indicam que a solução avaliada está próxima da melhor solução conhecida. A proposição contrária também é verdadeira.

4.1. Instâncias do Problema

As instâncias de teste foram geradas da seguinte forma. Dois grupos foram criados de acordo com os fatores n (número de tarefas), m (número de máquinas) e F (número de famílias). Instâncias pequenas: $n \in \{10, 15, 20\}$, $m \in \{2, 4, 6\}$ e $F \in \{2, 3\}$. Instâncias grandes: $n \in \{30, 40, 50\}$, $m \in \{2, 4, 6\}$ e $F \in \{4, 5\}$. Os tempos de processamento p_{jk} de cada tarefa j em cada máquina k foram gerados de três maneiras:

1. Tarefas correlacionadas. Atribui-se a cada tarefa j um tempo p_j retirado de uma distribuição uniforme $U(1, 100)$ mais um tempo adicional $p'_{jk} \in U(1, 20)$ por máquina. Deste modo, $p_{jk} = p_j + p'_{jk}$.
2. Máquinas correlacionadas. Atribui-se a cada máquina k um tempo $p_k \in U(1, 100)$ mais um tempo adicional $p'_{jk} \in U(1, 20)$ por tarefa. Deste modo, $p_{jk} = p_k + p'_{jk}$.
3. $p_{jk} \in U(10, 100)$.

Os tempos de preparação para cada família f foram retirados de $U(10, 100)$. Datas de entrega foram calculadas de acordo com [Schaller, 2014]. Para cada tarefa j, d_j é retirada de uma distribuição uniforme com limites $P \times (1 - R_1 - R_2)$ e $P \times (1 - R_1 + R_2)$, onde $P = \frac{\sum_j \sum_k p_{jk} + \sum_i \sum_j \sum_k s_{ijk}}{m^2}$. $R_1, R_2 \in \{0.25, 0.50\}$ determinam a urgência de entrega e o espaçamento entre datas, respectivamente.

As demandas de recurso foram definidas como: $q_j \in U(1, 20), \forall j \in J$. O inventário inicial r_k^* de cada máquina k, bem como suas taxas de incremento de recursos r_k seguiram as diretrizes de [Herr e Goel, 2016]. Duas sequências de tarefas S_1 e S_2 são geradas aleatoriamente. As tarefas de S_1 são agrupadas por famílias. As tarefas de S_2 são ordenadas ascendentemente pelas datas de entrega. Em cada máquina k, são calculados o *makespan* de cada sequência desconsiderando restrições de recurso. Sejam t_{1k} e t_{2k} os *makespan* das sequências S_1 e S_2 em cada máquina k, temos $t_{1k} < t_{2k}$, pois mais tempos de preparação são necessários em S_2. Consequentemente, $r'_k = \frac{2}{t_{1k}+t_{2k}} \sum_j q_j$ é a inclinação da reta média entre as retas de demandas das sequências S_1 e S_2 em cada máquina k. Como r'_k é calculado considerando todas as tarefas $j \in J$ alocadas em k, fazemos $r_k = r'_k/m$. O inventário inicial r_k^* de cada máquina k é definido como $r_k^* = 0.70 \times \left[\frac{\max_{\forall j}\{Q_j^{(1)} - r'_k C_{jk}\}}{m}\right]$, onde $Q_j^{(1)}$ é a demanda cumulativa de recursos de todas as tarefas em S_1 até uma tarefa j.

A associação dos fatores n, m, F, p_{jk}, R_1 e R_2 leva a 216 combinações possíveis para cada grupo de instâncias. Para cada combinação foram geradas duas instâncias, dessa forma, cada grupo possui 432 instâncias. Portanto, neste trabalho foram geradas 864 instâncias-problema.

4.2. Parâmetros de Inicialização dos Algoritmos

Os algoritmos descritos na Seção 3 dependem da entrada de parâmetros de inicialização. Para obter a melhor performance de cada método de resolução, realizamos uma etapa de calibração desses parâmetros em um subconjunto das instâncias.

4.2.1. *Parâmetros do Simulated Annealing*

Os parâmetros de inicialização do Algoritmo 1 (SA) são: α (fator de arrefecimento); e β (fator de parada). O conjunto dos valores testados foram: $\alpha \in \{0.70, 0.80, 0.90, 0.95\}$ e $\beta \in \{0.5, 1.0, 1.5, 2.0\}$. Os parâmetros do Algoritmo 2 (determinação da temperatura inicial) foram fixados com valores $\gamma = 0.95$, $\delta = 1.1$ e $T_{min} = 1.0$. A combinação dos valores α e β resultaram em 16 configurações. Cada instância-problema foi resolvida 5 vezes por todas as 16 configurações. A qualidade das soluções foi avaliada de acordo com o RDI (Equação 17). Os resultados experimentais foram analisados pelo teste estatístico não-paramétrico de Kruskal-Wallis. Neste teste, duas hipóteses são avaliadas: (i) hipótese nula: todos os métodos são estatisticamente equivalentes; (ii) hipótese alternativa: pelo menos uma das configurações é estatisticamente diferente. A Figura 1 mostra as médias e intervalos de confiança com 95% de confiabilidade estatística. Nessa figura, intervalos que se sobrepõem indicam que não há diferença significativa entre as configurações sobrepostas. O valor-P $= 0.00 \leq 0.05$ calculado no teste estatístico indica que a hipótese nula pode ser rejeitada. Podemos observar que os resultados melhoram à medida que os valores α e β aumentam. A configuração ($\alpha = 0.95$, $\beta = 2.0$) apresenta o menor RDI médio. Como os valores dessa configuração são valores-limite dos conjuntos testados, foi realizado mais um experimento de calibração. Os novos valores analisados foram: $\alpha \in \{0.95, 0.99\}$; $\beta \in \{2.0, 2.5, 3.0, 3.5, 4.0\}$. Os resultados obtidos são mostrados na Figura 2. A configuração ($\alpha = 0.99$, $\beta = 3.5$) apresenta o menor RDI médio. Essa configuração será usada posteriormente para análise comparativa com os métodos CPLEX e IG.

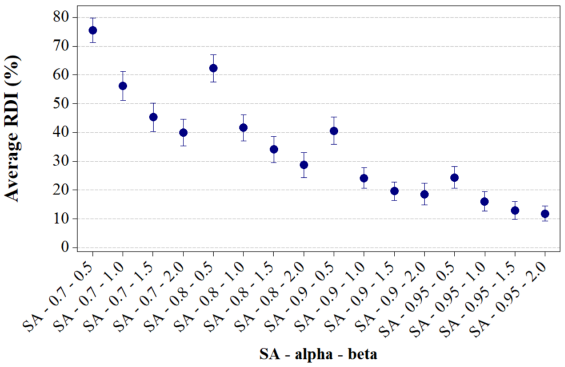


Figura 1: Gráfico de médias e intervalos de confiança com 95% de confiabilidade para o primeiro experimento de calibração.

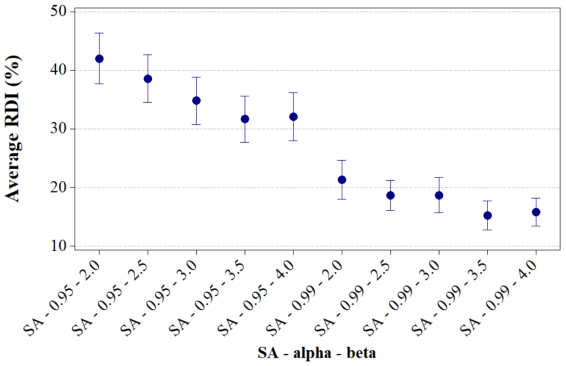


Figura 2: Gráfico de médias e intervalos de confiança com 95% de confiabilidade para o segundo experimento de calibração.

4.2.2. *Parâmetros do Iterated Greedy*

O Algoritmo 3 (IG) depende dos parâmetros: α (proporção de tarefas removidas na fase de destruição); k_{max} (número máximo de iterações consecutivas sem melhora na solução). Os valores testados foram: $k_{max} \in \{100, 200, 300, 400, 500\}$; $\alpha \in \{0.05, 0.10, 0.15, 0.20, 0.25, 0.30\}$. A combinação dos valores α e β resultaram em 30 configurações. Os resultados experimentais foram analisados pelo teste estatístico de Kruskal-Wallis. O valor-P $= 0.00 \leq 0.05$ calculado indica que a hipótese nula pode ser rejeitada. A Figura 3 mostra as médias e os intervalos de confiança com 95% de confiabilidade para as configurações testadas. A qualidade das soluções do algoritmo IG melhora à medida que o valor k_{max} aumenta. Entre as configurações testadas, ($k_{max} = 500$, $\alpha = 0.15$) apresenta o menor RDI médio e, portanto, essa configuração será utilizada nos próximos experimentos.

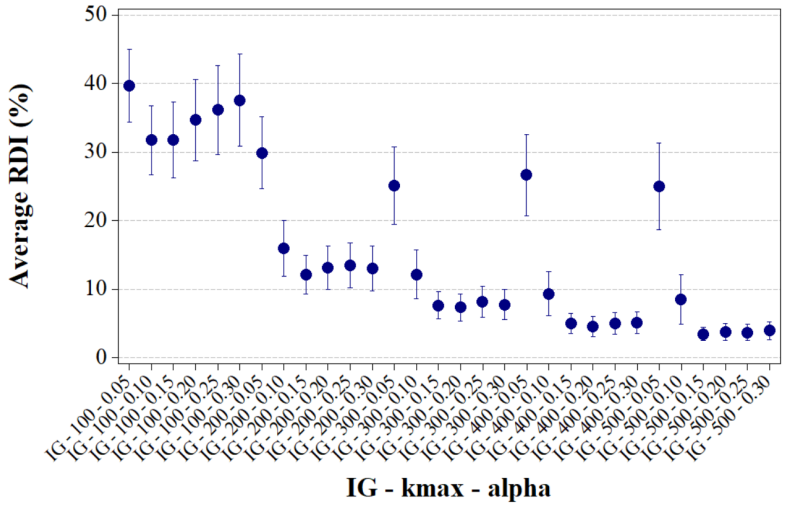


Figura 3: Gráfico de médias e intervalos de confiança com 95% de confiabilidade para o experimento de calibração.

5. Resultados e Análise de Resultados

Nesta seção apresentamos e discutimos os resultados do experimentos conduzidos a fim de avaliar a eficácia dos métodos de resolução sobre as 864 instâncias geradas neste trabalho. O modelo matemático foi resolvido pelo *solver* CPLEX dentro de um tempo limite de 3600 segundos. Como há a possibilidade de resolução incompleta neste tempo, a solução retornada é a melhor solução corrente encontrada. Os algoritmos SA e IG são de natureza estocástica e, por isso, cada instância-problema foi resolvida 10 vezes para que a análise dos resultados fosse representativa. A qualidade das soluções encontradas foi avaliada pelo RDI (Equação 17), onde f_{best} e f_{worst} são a melhor e a pior solução, respectivamente, encontradas dentre todas as execuções dos algoritmos. Para efeito comparativo, os RDIs calculados para SA e IG utilizaram como valores f_{best} tanto o valor médio, quanto o menor valor de atraso encontrado nas resoluções.

A comparação dos resultados das execuções do CPLEX, SA e IG são mostrados na Tabela 1. Os valores foram agrupados por número de tarefas (n) e por número de máquinas (m). O método exato de resolução CPLEX não foi capaz de encontrar soluções viáveis em tempo hábil para instâncias com $n > 20$ devido ao grande número de variáveis do modelo. Os resultados do CPLEX pioram consideravelmente à medida que o número de tarefas aumenta, tornando-o ineficaz (soluções de baixa qualidade) e ineficiente (tempos muito altos) para a resolução do problema. Deste modo, para a resolução de instâncias com $n = 30$, $n = 40$ e $n = 50$, o método CPLEX é considerado inapropriado, pois não é capaz de retornar soluções viáveis dentro do tempo limite estipulado. Para todos os grupos de instâncias, é notável a celeridade do algoritmo SA. Seus tempos de execução agigantam os tempos observados em CPLEX e IG. A qualidade mediana das soluções retornadas é, contudo, explicado pela ausência de uma fase interna de melhoria (como uma busca local, por exemplo). O algoritmo SA explora o espaço de soluções sem intensificar a busca. Consequentemente, o tempo de execução desse método é menor. O algoritmo IG apresentou, indiscutivelmente, as soluções de melhor qualidade. O sucesso desse algoritmo tem como causas: as fases de destruição e reconstrução das soluções, que exploram largamente o espaço de busca; a fase interna de melhoria (busca local), que explora intensivamente as vizinhanças locais das soluções; e a exaustão de tentativas de melhorias de soluções, que prolonga o tempo de execução do algoritmo caso detecte uma nova oportunidade de progresso.

A Figura 4 apresenta o desempenho médio dos algoritmos SA e IG de acordo com as diferentes organizações dos tempos de processamento p_{jk}. O modo como esses tempos são dispostos não interferem no desempenho dos métodos, pois estes apresentam padrões recorrentes de comportamento. Pela análise da Tabela 1 e dos padrões gráficos da Figura 4 concluímos que: (i) o desempenho do IG decresce à medida que n cresce; (ii) o desempenho do SA varia entre os grupos, porém oscila pouco dentro deles. A Figura 5 apresenta o gráfico das médias e intervalos de confiança para os métodos de resolução em diferentes grupos de instâncias. Foi aplicado o teste de Kruskal-Wallis para avaliar as diferenças entre as implementações. Nitidamente, a superioridade do método IG é observada: os intervalos de confiança não se sobrepõem com os intervalos de nenhum outro método; os RDIs médios são os menores. Para instâncias pequenas, SA e IG são estatisticamente superiores ao CPLEX. Para instâncias maiores, o algoritmo IG é estatisticamente superior ao SA.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

Tabela 1: RDIs (%) e tempos de CPU (em segundos) para as resoluções de CPLEX, SA e IG.

		CPLEX		SA			IG		
n	m		Time	Best	Avg	Time	Best	Avg.	Time
10	2	0.00	1,421.89	1.22	19.72	0.25	0.00	1.49	1.38
	4	2.44	2,917.96	4.45	38.08	0.31	0.00	1.61	1.77
	6	0.00	1,216.17	0.72	24.04	0.38	0.00	0.23	2.12
15	2	59.42	3,362.85	8.52	33.97	0.44	0.00	1.96	1.87
	4	85.23	3,571.31	8.45	40.83	0.55	0.00	2.77	2.86
	6	89.76	3,554.39	5.85	29.62	0.67	0.00	1.53	3.44
20	2	82.06	3,432.82	5.04	21.72	0.67	0.00	0.69	2.65
	4	96.71	3,561.97	11.37	26.02	0.84	0.00	1.91	4.65
	6	99.89	3,543.54	5.38	17.70	1.00	0.00	2.02	5.83
30	2	-	-	14.68	47.85	1.28	0.00	2.85	5.40
	4	-	-	20.26	58.43	1.55	0.37	9.95	8.75
	6	-	-	19.23	56.51	1.83	2.40	13.86	11.14
40	2	-	-	11.64	40.90	2.02	0.71	7.14	7.87
	4	-	-	18.90	53.76	2.44	1.62	16.73	13.28
	6	-	-	15.60	55.98	2.84	4.11	21.73	15.65
50	2	-	-	12.49	49.32	2.99	2.19	14.43	12.90
	4	-	-	15.27	51.02	3.50	7.51	25.79	17.90
	6	-	-	9.71	52.21	4.04	9.90	32.46	22.52

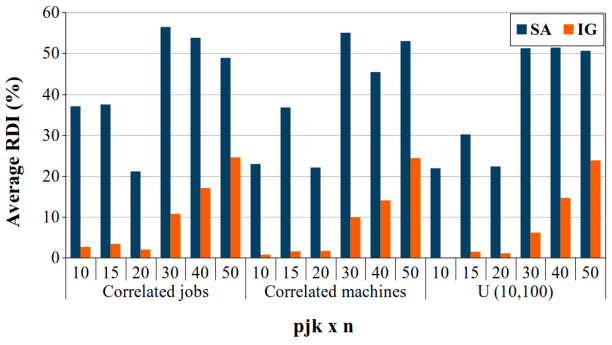


Figura 4: RDIs médios (%) em instâncias por $p_{jk} \times n$

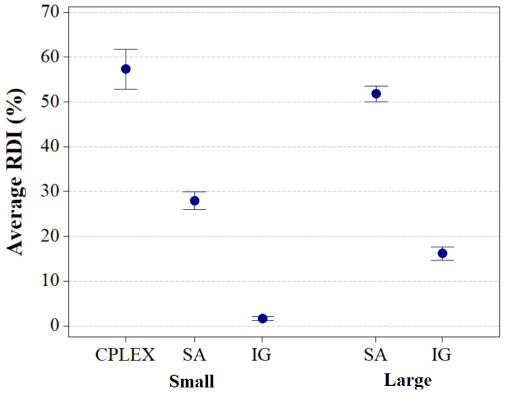


Figura 5: Gráfico de médias e intervalos de confiança com 95% de confiabilidade para os algoritmos por grupos de instâncias

6. Conclusões e Estudos Futuros

Neste trabalho buscamos apresentar e resolver o problema da programação de tarefas em máquinas paralelas não-relacionadas sujeitas a tempos de preparação e restrições de recurso para minimização do atraso total. Para o problema em questão foi desenvolvido uma formulação matemática com programação linear inteira-mista. Por se tratar de um problema NP-difícil, métodos exatos de resolução são conhecidamente ineficientes. Deste modo, propomos dois métodos resolutivos baseados nas meta-heurísticas *simulated annealing* (SA) e *iterated greedy* (IG). Um con-

junto de instâncias-problema foi gerado neste trabalho para analisar o desempenho dos métodos propostos. O modelo matemático foi resolvido pelo *solver* CPLEX, e os parâmetros dos métodos heurísticos SA e IG foram cuidadosamente calibrados. Experimentos computacionais demonstraram a superioridade do método IG para encontrar soluções para o problema. Todos os resultados foram estatisticamente testados. Futuramente, o algoritmo IG pode ser aplicado em problemas siderúrgicos concretos com dados reais de produção. Ademais, outras funções objetivo podem ser incorporadas ao o problema, como o *makespan* e, dessa forma, conceber um problema de otimização multi-objetivo.

Agradecimentos

Os autores agradecem o apoio financeiro das agências de fomento CNPq e CAPES.

Referências

Aarts, E., Korst, J., e Michiels, W. (2005). Simulated annealing. In *Search methodologies*, p. 187–210. Springer.

Arroyo, J. E. C., Leung, J. Y.-T., e Tavares, R. G. (2019). An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times. *Engineering Applications of Artificial Intelligence*, 77:239–254.

Du, J. e Leung, J. Y.-T. (1990). Minimizing total tardiness on one machine is np-hard. *Mathematics of operations research*, 15(3):483–495.

Fanjul-Peyro, L. e Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69.

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.

Jiang, S.-L., Zheng, Z., e Liu, M. (2018). A preference-inspired multi-objective soft scheduling algorithm for the practical steelmaking-continuous casting production. *Computers & Industrial Engineering*, 115:582–594.

Pinheiro, J. C. e Arroyo, J. E. C. (2020). Effective ig heuristics for a single-machine scheduling problem with family setups and resource constraints. *Annals of Mathematics and Artificial Intelligence*, 88(1-3):169–185.

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.

Schaller, J. E. (2014). Minimizing total tardiness for scheduling identical parallel machines with family setups. *Computers & Industrial Engineering*, 72:274–281.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings