Um algoritmo heurístico para o problema de escalonamento de lotes com penalidades por atraso

Eduardo Queiroga
Instituto de Computação - Universidade Federal Fluminense - UFF
Av. Gal. Milton Tavares de Souza, s/n, São Domingos — Niterói - RJ - Brasil, 24210-310
eduardoqueiroga@id.uff.br

Rian Pinheiro
Universidade Federal Rural de Pernambuco - Unidade Acadêmica de Garanhuns
Avenida Bom Pastor, s/n - Boa Vista, Garanhuns - PE, 55292-270
rian.gabriel@gmail.com

Quentin Christ
Ecole des Mines de Saint-Etienne - Department of Manufacturing Sciences and Logistics
quentin.grenoble@gmail.com

Anand Subramanian
Universidade Federal da Paraíba - Centro de Informática
R. dos Escoteiros, s/n - Mangabeira, João Pessoa - PB, 58055-000
anand@ci.ufpb.br

RESUMO

Este trabalho trata do problema de escalonamento de tarefas em uma máquina de processamento em lote com o objetivo de minimizar o total de penalidades por atraso. Esse problema emerge em várias aplicações práticas dentre as quais é possível citar a fabricação de materiais semicondutores. Para resolvê-lo, uma meta-heurística baseada em busca local com cinco estruturas de vizinhança é proposta e uma discussão detalhada acerca dos procedimentos de avaliação é realizada. Resultados computacionais são reportados para um novo conjunto de instâncias, com número de tarefas variando entre 10 e 50.

PALAVRAS CHAVE. Escalonamento em lotes; Penalidade por atraso; Busca local

Tópicos: MH - Meta-Heurísticas

ABSTRACT

This work concerns the single machine total weighted tardiness batch scheduling problem. This problem arises in several practical applications and among them, we can cite the manufacturing of semiconductors materials. To solve it, a local search-based metaheuristic with five neighborhood structures is proposed and a detailed discussion on the move evaluation procedures is conducted. Computational results are reported for newly generated benchmark instances ranging from 10 to 50 jobs.

KEYWORDS. Batch scheduling; Weighted tardiness; Local search

Paper topics: MH - Metaheuristics

https://proceedings.science/p/85378?lang=pt-br

1. Introdução

Seja $J = \{1, 2, \ldots, n\}$ um conjunto de tarefas tal que uma tarefa j possui os seguintes atributos: tempo de processamento p_j, tamanho s_j, data de liberação r_j (instante de tempo em que a tarefa poderá começar a ser processada), data de entrega d_j e penalidade w_j para cada uma unidade de tempo de atraso. Este artigo trata do problema de escalonar essas tarefas em lotes para processamento em uma máquina de tal forma que o atraso penalizado total seja mínimo. Especificamente, o problema envolve duas decisões principais: particionar as tarefas em grupos ou lotes (cada tarefa deve estar exatamente em um dos lotes) e sequenciar esses lotes. A separação em lotes modela o cenário onde tarefas são processadas em conjunto e, portanto, um lote B_i só pode começar a ser processado no instante $r(B_i) = \max_{j \in B_i} r_j$ com tempo de processamento $p(B_i) = \max_{j \in B_i} p_j$. Dada uma sequência de lotes $\mathcal{B} = \{B_1, B_2, \ldots B_m\}$, $C(B_1) = r(B_1) + p(B_1)$ é o tempo de término do primeiro lote B_1 e $C(B_i) = \max\{r(B_i) + p(B_i), C(B_{i-1})\}$ é o tempo de término do lote B_i, para $i \in \{2, 3, \ldots, m\}$. Os lotes possuem uma capacidade Q que deve ser respeitada, ou seja, $s(B_i) = \sum_{j \in B_i} s_j \leq Q$ para cada lote B_i. O atraso de uma tarefa j é dado por $T_j = \max\{0, C_j - d_j\}$, tal que C_j é o tempo de término da tarefa (que é igual ao tempo de término do lote que possui j). O objetivo do problema é encontrar uma solução de modo que o atraso penalizado total $\sum_{j \in J} w_j T_j$ seja mínimo. Esse problema também pode ser referido por meio da notação $1|batch, r_j, s_j, compt| \sum w_j T_j$, que é bastante utilizada na literatura de escalonamento. Em Peter et al., o $1|batch, r_j, s_j, compt| \sum w_j T_j$ foi provado ser NP-Difícil no sentido forte, o que justifica o uso de abordagens heurísticas para resolução sem garantia de otimalidade. O termo $compt$ indica que qualquer par de tarefas pode estar no mesmo lote, em contraste com a versão do problema que considera famílias de tarefas incompatíveis [Balasubramanian et al., 2004; Perez et al., 2005]. A Tabela 1 apresenta uma instância do problema com 10 tarefas. Uma solução ótima para essa instância é ilustrada na Figura 1.

Problemas de escalonamento de tarefas em lotes têm sido intensamente estudados para aplicações na indústria manufatureira [Mathirajan e Sivakumar, 2006]. Em Lee et al. [1992] foi introduzida a aplicação de escalonamento em lotes para as operações de teste de materiais semicondutores na produção de dispositivos (e.g. microprocessadores e memórias DRAM). Nesse processo, um tipo de forno especial, conhecido como *burn-in oven*, é utilizado para testar a resistência dos dispositivos em altas temperaturas, detectando o que é conhecido como *mortalidade infantil*. Considerando o $1|batch, r_j, s_j, compt| \sum w_j T_j$, o conjunto de chips do mesmo tipo são colocados em uma ou mais prateleiras (definindo o tamanho da tarefa, que pode variar para cada tipo de chip) e o limite de prateleiras do forno caracteriza a restrição de capacidade de um lote. O tempo de processamento de cada tarefa é o tempo mínimo de teste que aquele tipo de chip deve ser exposto. O tempo de término das tarefas será definido pelo maior tempo de processamento, uma vez que o teste não pode ser interrompido até que a tarefa de maior tempo de processamento seja realizada (as outras tarefas podem ficar mais tempo do que o necessário). Minimizar o atraso penalizado total (ou outras funções objetivo, tal como minimizar o *makespan* [Xu et al., 2012] ou *flow-time* [Parsa et al., 2016]) podem reduzir significativamente os custos envolvidos no processo de produção desse tipo de material, principalmente por se tratar de um teste de longa duração quando comparado a outras operações do processo. A vantagem da função objetivo tratada sobre outras é que em muitos cenários, a indústria requer métricas com foco na data de entrega das atividades.

A partir do levantamento da literatura, pode-se afirmar que os seguintes trabalhos trataram do mesmo problema abordado neste artigo. Em Chou e Wang [2008], uma formulação de programação inteira é proposta além de duas heurísticas híbridas com um algoritmo de programação dinâmica. Os melhores resultados foram obtidos pela heurística baseada em Algoritmo Genético (AG), quando comparada com a outra heurística e o modelo matemático (limitado por tempo). Mathirajan et al. [2010] propuseram uma heurística de construção gulosa e uma meta-heurística baseada no *Simulated Annealing* (SA), que foi avaliada com relação a uma formulação matemática.

https://proceedings.science/p/85378?lang=pt-br

Tarefa	p_j	d_j	s_j	w_j	r_j
1	9	12	4	1	1
2	6	11	6	8	3
3	12	16	8	2	0
4	15	24	4	6	2
5	8	14	8	7	0
6	17	25	4	9	4
7	20	44	4	8	4
8	15	22	5	2	1
9	12	17	6	1	2
10	6	25	6	1	0

Tabela 1: Instância do problema com 10 tarefas.

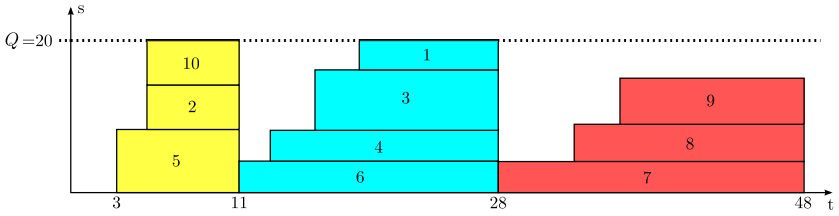


Figura 1: Representação gráfica da instância descrita na Tabela 1. A solução $\mathcal{B} = \{\{5, 2, 10\}, \{6, 4, 3, 1\}, \{7, 8, 9\}\}$ é ótima e possui penalidade por atraso dada por $\sum_{j \in J} w_j T_j = (w_5 T_5 + w_2 T_2 + w_{10} T_{10}) + (w_6 T_6 + w_4 T_4 + w_3 T_3 + w_1 T_1) + (w_7 T_7 + w_8 T_8 + w_5 T_9) = 206$.

Wang [2011] propôs uma heurística de duas fases, sendo a primeira fase um método populacional responsável por definir uma sequência de tarefas a ser loteada por uma algoritmo de programação dinâmica (segunda fase). Além disso, um método iterativo foi proposto para melhorar a solução obtida pela heurística. Com o método iterativo, foram obtidos resultados de melhor qualidade quando comparado ao método proposto por Chou e Wang [2008]. Vélez-Gallego et al. [2011] propuseram uma meta-heurística *Variable Neighborhood Search* (VNS) que se mostrou superior ao AG de Chou e Wang [2008] com relação as soluções ótimas para um conjunto de instâncias pequenas (até 15 tarefas). Zinouri et al. [2012] propuserem uma meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) que foi comparada com o VNS de Vélez-Gallego et al. [2011] e o AG de Chou e Wang [2008] para o mesmo conjunto de instâncias considerado em Vélez-Gallego et al. [2011].

O objetivo deste trabalho é desenvolver uma simples metodologia de busca local baseada em ILS e VND que seja capaz de resolver eficientemente o $1|batch, r_j, s_j, compt| \sum w_j T_j$. Um conjunto de vizinhanças conhecidas foi utilizado e uma discussão detalhada sobre a avaliação de movimentos para duas das vizinhanças foi realizada. Além disso, um novo conjunto de instâncias de teste foi gerado com o objetivo de superar fragilidades dos *benchmarks* da literatura e gerar instâncias mais difíceis. Em comparação com limitantes para o problema, o método proposto se mostrou capaz de encontrar soluções de alta qualidade em baixo tempo computacional.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta em detalhes a metodologia proposta para resolução do $1|batch, r_j, s_j, compt| \sum w_j T_j$. A Seção 3 apresenta o projeto de instâncias para o problema, bem como a análise dos resultados obtidos pelos experimentos computacionais realizados. Finalmente, na Seção 4 as principais conclusões obtidas são apresentadas bem como sugestões para trabalhos futuros.

2. Algoritmo proposto

O algoritmo proposto ILS-Batch consiste em uma adaptação da meta-heurística *Iterated Local Search* (ILS) [Lourenço et al., 2010] juntamente com um procedimento *Variable Neighborhood Descent* [Hansen et al., 2010], com uma ordem de vizinhança definida aleatoriamente na fase de busca local. Métodos híbridos baseados em adaptações de meta-heurísticas populares na literatura têm apresentado boa performance em termos de qualidade de solução [Talbi, 2013]. Da

https://proceedings.science/p/85378?lang=pt-br

mesma forma, abordagens baseadas em ILS e VND têm sido aplicadas com sucesso em diferentes problemas conhecidos, como por exemplo: roteamento de veículos [Penna et al., 2013; Subramanian et al., 2013], escalonamento [Subramanian e Farias, 2017] e formação de células de manufatura [Martins et al., 2015].

O pseudocódigo do ILS-Batch é apresentado no Algoritmo 1. O algoritmo tem como parâmetros de entrada: (i) I_R que representa o número de iterações *multi-start* do ILS-Batch; (ii) I_{ILS} que define a quantidade de iterações sem melhoramentos antes de reiniciar o *multi-start* e (iii) α que contabiliza a quantidade de elementos considerados pelo método construtivo HG_{EDD} na linha 4.

O método construtivo HG_{EDD} é baseado no critério *Earliest Due Date* (EDD), bastante conhecido na literatura de escalonamento [Vepsalainen, 1987]. Basicamente, as tarefas são ordenadas de forma não-decrescente com respeito a data de entrega d_j e uma lista de lotes já construídos é mantida (inicialmente vazia). Uma das α primeiras tarefas da sequência é escolhida aleatoriamente (com distribuição uniforme) para entrar no lote atualmente em construção. Se a tarefa escolhida não couber no lote, tenta-se colocar a menor tarefa (com menor s_j) entre as α tarefas. Se uma inserção não for possível, o lote é adicionado no fim da lista de lotes e um novo lote vazio é criado para ser preenchido. Quando uma tarefa é adicionada a um lote, ela é removida da sequência de tarefas. O método é executado até que todas as tarefas sejam adicionadas a algum lote.

A busca local (linha 8) — VND com ordem aleatória de exploração das vizinhanças — utiliza o mecanismo *best improvement* para definir o movimento realizado para a vizinhança escolhida. Esse mecanismo avalia todas as possibilidades de movimento (soluções vizinhas) de uma determinada estrutura de vizinhança para realizar o que produz o melhor ganho (se houver) na função objetivo. Finalmente, a linha 12 executa a perturbação.

Algoritmo 1: ILS-Batch

```
1  Procedimento ILS-Batch(J, Q, I_R, I_ILS, α)
2      f* ← ∞
3      para iter = 1 ... I_R faça
4          B ← HG_EDD(J, Q, α)
5          B' ← B
6          iter_ILS ← 0
7          enquanto iter_ILS ≤ I_ILS faça
8              B ← BuscaLocal(B);
9              se f(B) < f(B') então
10                 B' ← B
11                 iter_ILS ← 0
12             B ←Perturba(B')
13             iter_ILS ← iter_ILS + 1
14         se f(B') < f* então
15             B* ← B'
16             f* ← f(B')
17     retorne B*
```

2.1. Estruturas de vizinhança

Nesta seção, são descritas as estruturas de vizinhanças utilizadas na metodologia de busca local proposta. Para as estruturas *Insertion* e *Swap*, é discutido em detalhes uma avaliação do movimento que tira proveito da manutenção dos tempos de término dos lotes.

2.1.1. Inserção de tarefas (*Insertion*):

Dada uma solução $\mathcal{B}$, essa estrutura define um conjunto de soluções vizinhas que pode ser obtido por todas as possibilidades de remoção de uma tarefa j em um lote B_k e inserção dessa

https://proceedings.science/p/85378?lang=pt-br

tarefa em outro lote B_l, tal que $k \neq l$. Apenas movimentos viáveis podem ser considerados, ou seja, se $s_j + s(B_l) \leq Q$. Obviamente, se uma das condições (1)–(2) for satisfeita, as inserções de j em lotes subsequentes a B_k não precisam ser avaliadas pois sabe-se de antemão que tais movimentos não levarão a uma melhora.

$$p_j < p(B_k) \tag{1}$$

$$p_j = p(B_k) \text{ e } \exists q \in B_k \setminus \{j\} \text{ tal que } p_q = p_j \tag{2}$$

Na Figura 2, um exemplo de *Insertion* é apresentado. É possível notar que a inserção da tarefa 2 no lote B_3 permite que todos os lotes tenham seus tempos de término reduzidos, o que tende a gerar melhoramentos, principalmente se d_2 for alto e/ou w_2 for baixo.



(a) (b)

Figura 2: Exemplo de solução gerada por um movimento de *Insertion* para a solução $\mathcal{B} = \{B_1 = \{1, 2\}, B_2 = \{4, 3\}, B_3 = \{5\}\}$. O movimento se dá pela remoção da tarefa 2 do lote B_1 na solução em (a) e pela inserção de 2 no lote B_3, como ilustrado em (b).

O Algoritmo 2 avalia o ganho (ou perda) na função objetivo para um determinado movimento sem a necessidade de realizá-lo de fato. A eficiência do algoritmo depende dos novos tempos de término dos lotes. Seja $c'(B_e)$ o novo tempo de término do lote B_e (declarado na linha 4). Esse valor pode ser calculado em tempo $\mathcal{O}(1)$, através da expressão $c'(B_e) = \max\{c(B_{e-1}), r'(B_e)\} + p'(B_e)$, onde $r'(B_e)$ e $p'(B_e)$ denotam o data de liberação e tempo de processamento do lote, respectivamente, após o movimento. O cálculo de $c'(B_d)$ depende da mudança no tempo de término dos lotes anteriores e pode ser computado em tempo $\mathcal{O}(|\mathcal{B}|)$ no pior caso. O melhor caso do algoritmo configura-se quando os tempos de termino de B_e e B_d não se alteram após o movimento e a variação de atraso penalizado da tarefa j determina o ganho total (linha 5). Porém, quando ambos os tempos de término são alterados, caracterizando o pior caso, todas as tarefas de lotes a partir de B_e devem ser analisadas (linha 14).

Algoritmo 2: Avaliação do movimento de inserção de $j \in B_k$ em B_l

```
1  Procedimento CompImpInsertion(B, k, j, l)
2      B_e é o lote mais a esquerda entre B_k e B_l em B, ou seja, e = min{k, l}
3      B_d é o lote mais a direita entre B_k e B_l em B, ou seja, d = max{k, l}
4      c'(B_e) e c'(B_d) são os tempos de término de B_d e B_e após o movimento, respectivamente
5      Δ ← (w_j × max{0, c(B_k) − d_j}) − (w_j × max{0, c'(B_l) − d_j})
6      se c'(B_e) = c(B_e) então
7          se c'(B_d) ≠ c(B_d) então
8              para i = d, d + 1, ..., |B| faça
9                  para cada q ∈ B_i faça
10                     se q ≠ j então
11                         c'(B_i) = max{c'(B_{i−1}), r'(B_i)} + p'(B_i)
12                         Δ ← Δ + (w_q × max{0, c(B_i) − d_q}) − (w_q × max{0, c'(B_i) − d_q})
13     senão
14         Atualiza Δ para cada tarefa (exceto j) em todos os lotes a partir de B_e (análogo ao laço na linha 8),
           utilizando c'(B_{i−1}) para definir c'(B_i)
15     retorna Δ
```

É importante observar que o laço da linha 8 pode ser interrompido quando $c'(B_i) = c(B_i)$, uma vez que todas as penalidades daquele lote em diante não serão alteradas. Algo semelhante pode

https://proceedings.science/p/85378?lang=pt-br

ser feito na linha 14 para lotes entre B_e e B_d, sendo possível ir diretamente (após o primeiro lote com tempo de término inalterado) para o cálculo de $c'(B_d)$. Além disso, essa condição pode ser utilizada para encerrar a execução para lotes após B_d.

2.1.2. Troca de tarefas (*Swap*):

Dada uma solução $\mathcal{B}$, essa estrutura define um conjunto de soluções vizinhas que pode ser obtido por todas as permutações (trocas) entre duas tarefas de lotes distintos. Formalmente, são consideradas as trocas entre i e j, $\forall i \in B_k, \forall j \in B_l, \forall k, l \in \{1, 2, \ldots, |\mathcal{B}|\}$, tal que $k < l$. Como no *Insertion*, apenas movimentos que respeitam a capacidade podem ser considerados, ou seja, se $(s_j + s(B_k) - s_i) \leq Q$ e $(s_i + s(B_l) - s_j) \leq Q$.

Na Figura 3, um exemplo de *Swap* é apresentado. De forma análoga ao *Insertion*, esse movimento pode reduzir os tempos de término de vários lotes (no exemplo, isso acontece para todos os lotes, como mostram as Figuras 3(a) e 3(b)) ou simplesmente induzir uma melhora pela diferença de penalidades geradas pelas duas tarefas envolvidas, mesmo que os tempos de término dos lotes sejam os mesmos. A avaliação de um movimento de *Swap* segue o mesmo princípio do *Insertion* e é apresentada no Algoritmo 3.



(a) (b)

Figura 3: Exemplo de solução gerada por um movimento de *Swap* para a solução $\mathcal{B} = \{B_1 = \{1, 2\}, B_2 = \{4, 3\}, B_3 = \{5\}\}$. O movimento se dá pela troca entre tarefas $2 \in B_1$ e $3 \in B_2$ na solução em (a), resultando na solução ilustrada em (b).

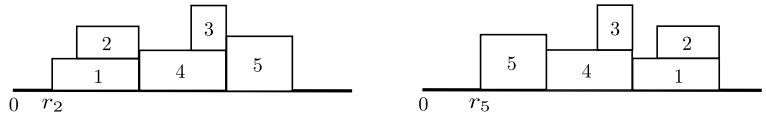
Algoritmo 3: Avaliação do movimento de troca de tarefas $i \in B_k$ e $j \in B_l$

```
1  Procedimento CompImpSwap(B, k, i, l, j)
2      c'(B_k) e c'(B_l) são os tempos de término de B_k e B_l após o movimento, respectivamente
3      Δ ← 0
4      se c'(B_k) = c(B_k) então
5          Δ ← Δ + (w_j × max{0, c(B_l) − d_j} − w_j × max{0, c'(B_k) − d_j})
6          Δ ← Δ + (w_i × max{0, c(B_k) − d_i} − w_i × max{0, c'(B_l) − d_i})
7          se c'(B_l) ≠ c(B_l) então
8              para i = l, l + 1, ..., |B| faça
9                  para cada q ∈ B_i faça
10                     se q ≠ j então      /* quando B_i = B_l, ignora-se j que irá sair */
11                         c'(B_i) = max{c'(B_{i−1}), r'(B_i)} + p'(B_i)
12                         Δ ← Δ + (w_q × max{0, c(B_i) − d_q}) − (w_q × max{0, c'(B_i) − d_q})
13     senão
14         Atualiza Δ para cada tarefa (considerando j em B_k e i em B_l) em todos os lotes a partir de B_k
             (análogo ao laço na linha 8), utilizando c'(B_{i−1}) para definir c'(B_i)
15     retorna Δ
```

2.1.3. Troca de lotes (*BatchSwap*):

Essa estrutura define um conjunto de soluções vizinhas que pode ser obtido por todas as permutações entre dois lotes distintos B_k e B_l na sequência $\mathcal{B}$, tal que $k, l \in \{1, 2, \ldots, |\mathcal{B}|\}$ e $k < l$. Mais formalmente, o movimento entre B_k e B_l transforma $\mathcal{B} = \{\ldots, B_k, \ldots, B_l, \ldots\}$ em $\mathcal{B}' = \{\ldots, B_l, \ldots, B_k, \ldots\}$. Na Figura 4, um exemplo de *BatchSwap* é apresentado. Assim como as estruturas anteriores, as mudanças produzidas nos tempos de término dos lotes podem acarretar em ganhos significativos na solução.

https://proceedings.science/p/85378?lang=pt-br



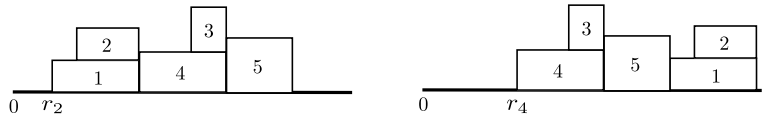
(a) (b)

Figura 4: Exemplo de solução gerada por um movimento de *BatchSwap* para a solução $\mathcal{B} = \{B_1 = \{1, 2\}, B_2 = \{4, 3\}, B_3 = \{5\}\}$. O movimento se dá pela troca entre os lotes B_1 e B_3 na solução em (a), resultando na solução ilustrada em (b).

2.1.4. Inserção de lotes (*BatchInsertion*):

Essa estrutura define um conjunto de soluções vizinhas obtido através de todos os reposicionamentos (ou inserções) de um lote B_k na sequência $\mathcal{B}$, $\forall k \in \{1, 2, \ldots, |\mathcal{B}|\}$. Mais formalmente, o lote B_k deve ser reinserido na posição k', tal que $k \neq k'$. Se $k < k'$, B_k será inserido após $B_{k'}$ (lembre-se que B'_k está na posição k' na solução inicial) e antes de $B_{k'+1}$ (se existir). Se $k > k'$, B_k será inserido antes de $B_{k'}$ e após $B_{k'-1}$ (se existir). Obrigatoriamente B_k tem que avançar (ou recuar) pelo menos duas posições na sequência, uma vez que a permutação com um lote vizinho pode ser alcançada através da vizinhança *BatchSwap*.

Na Figura 5, um exemplo dessa estrutura é apresentado. É possível notar que a solução resultante na Figura 5(b) não pode ser obtida por um único movimento *BatchSwap*, bem como a solução da Figura 4 (b) não pode ser obtida por um único *Insertion*. Essa é uma característica positiva, uma vez que é desejável que as vizinhanças tenham a menor intercessão possível.

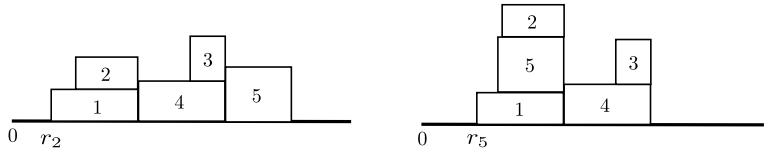


(a) (b)

Figura 5: Exemplo de solução gerada por um movimento de *BatchInsertion* para a solução $\mathcal{B} = \{B_1 = \{1, 2\}, B_2 = \{4, 3\}, B_3 = \{5\}\}$. O movimento se dá pela inserção do lote B_1 após o lote B_3 na solução em (a), resultando na solução ilustrada em (b).

2.1.5. União de lotes (*Merge*):

Essa estrutura é caracterizada pela transformação de dois lotes B_k e B_l em um único lote B_m que possui a união de todas as tarefas e é posicionado no lote mais a esquerda entre B_k e B_l. Essa operação é válida quando $\sum_{j \in B_k} s_j + \sum_{j \in B_l} s_j \leq Q$. A Figura 6 ilustra um movimento dessa vizinhança.



(a) (b)

Figura 6: Exemplo de solução gerada por um movimento de *Merge* para a solução $\mathcal{B} = \{B_1 = \{1, 2\}, B_2 = \{4, 3\}, B_3 = \{5\}\}$. O movimento se dá pela união dos lotes B_1 e B_3 na solução em (a), resultando na solução ilustrada em (b).

2.2. Perturbações

A perturbação é um mecanismo fundamental do ILS que tem como objetivo modificar uma solução de ótimo local a ponto de que a busca local consiga convergir para ótimos locais distintos. No ILS-Batch foram utilizadas duas perturbações:

Troca de tarefas (*Swap*): Baseada na vizinhança *Swap*, esta perturbação escolhe aleatoriamente dois lotes e uma tarefa de cada lote para realizar um movimento de *Swap*.

https://proceedings.science/p/85378?lang=pt-br

Divisão de lote (*Split*): Seleciona aleatoriamente um lote B_i e um ponto de corte $t \in [2, |B_i|-1]$. Considere que B_i está ordenado de forma não-crescente com relação a p_j. Em seguida, B_i é dividido em dois novos lotes B_i' e B_i'', onde B_i' recebe as primeiras t tarefas de B_i e B_i'' recebe as demais tarefas. Os lotes B_i' e B_i'' devem substituir, nesta ordem, B_i na sua posição na sequência de lotes.

A cada chamada de $Perturba$ no ILS-Batch, uma das duas perturbações é escolhida aleatoriamente e um ou dois movimentos (também definido aleatoriamente) são selecionados para alterar a solução. É possível que nenhum movimento seja realizado por serem inviáveis (devido a restrição de capacidade). Se isso acontecer, uma pertubação que permuta dois lotes vizinhos, escolhidos aleatoriamente, é realizada para evitar execuções desnecessárias da busca local.

3. Experimentos computacionais

Nesta seção, uma análise dos conjuntos de instâncias da literatura e o projeto de novas instâncias é discutido. Em seguida, as configurações do ambiente experimental e a parametrização do método proposto são apresentados. Finalmente, os resultados computacionais obtidos são reportados e discutidos.

3.1. Geração de instâncias

Não foi possível, por parte dos autores deste trabalho, obter acesso aos conjuntos de instâncias geradas em Chou e Wang [2008], Mathirajan et al. [2010] e Wang [2011] (mesmo procedimento de Mathirajan et al. [2010]). Apesar disso, um esforço para gerar instâncias de acordo com a descrição feita nos trabalhos foi realizado. Em ambos os modelos de geração de instâncias (Chou e Wang [2008], Mathirajan et al. [2010]), uma distribuição uniforme em um intervalo estabelecido foi considerada para os atributos do problema. Em Mathirajan et al. [2010], nenhuma informação foi dada a respeito da distribuição do parâmetro w_j e foi suposto que os autores consideraram $w_j = s_j$. Em Chou e Wang [2008], o intervalo dos tempos de entrega é baseado no *makespan* esperado C^*_{max} obtido pelo algoritmo FBLPT [Lee, 1999]. Com o C^*_{max} calculado, tem-se que d_j é distribuído no intervalo $I(R_1, T) = [u(1 - R_1/2); u(1 + R_1/2)]$, tal que $u = (1 - T)(\min(r_j) + C^*_{max})$, $T = \{0,3; 0,6\}$ e $R_1 = \{0,5; 2,5\}$. A distribuição dos parâmetros utilizada por ambos os autores está disponível na Tabela 2.

Foram realizados testes em instâncias geradas de acordo com o procedimento descrito anteriormente. Apesar de algumas instâncias apresentarem alguma dificuldade, certas observações devem ser feitas. Algumas instâncias demonstraram ser demasiadamente fáceis, principalmente para a configuração de Chou e Wang [2008]. Para problemas de 10 a 20 tarefas não é incomum encontrar soluções ótimas de custo 0, que podem ser facilmente resolvidas por buscas locais simples. Essa facilidade se justifica principalmente pela distribuição de d_j em valores muitos altos. Além disso, a definição de r_j tende a simplificar a resolução das instâncias. Nas duas configurações discutidas até aqui, o intervalo de r_j é fixo para qualquer tamanho de instância, sendo o valor máximo insuficiente para gerar instâncias de grande porte difíceis.

Dessa forma, foi projetado um conjunto de novas instâncias para o problema $1|batch, r_j, s_j, compt| \sum w_j T_j$ que explora as fragilidades discutidas e se propõe a ser mais desafiador para os algoritmos. A distribuição de parâmetros utilizada também é descrita na Tabela 2. Para cada configuração de parâmetro, foram geradas 5 instâncias, resultando em 810 instâncias diferentes. As instâncias foram divididas em grupos de 10 instâncias como resumido pela Tabela 3. Para cada grupo, 5 instâncias são com $Q = 20$ e as outras 5 com $Q = 40$.

3.2. Parametrização e configuração do ambiente experimental

A seguinte parametrização foi utilizada no ILS-Batch: $I_R = 20$ e $I_{ILS} = 4n$. Esses valores foram utilizados com sucesso em uma abordagem semelhante para um problema de escalonamento na literatura [Subramanian e Farias, 2017]. Além disso, valores muito próximos que os escolhidos para I_R e I_{ILS}, obtiveram bons resultados em uma classe de problemas de escalonamento [Kramer e Subramanian, 2017]. O ILS-Batch foi implementado em C++ e compilado através

https://proceedings.science/p/85378?lang=pt-br

Parâmetros	Níveis [Chou e Wang, 2008]	Níveis [Mathirajan et al., 2010]	Níveis Proposto
n	10,12,20,50,100	10,12,20,50,100	10,15,20,25,30,35,50,75,100
r_j	U(0,48)	U(1,20), U(1,30)	U(0, $R_2 \cdot C^*_{max}$)
p_j	U(8,48)	U(1,10), U(1,15)	U(1,30)
d_j	$r_j + p_j + I(R_1, T)$	$r_j + p_j$ + U(1,20), U(1,30)	$r_j + p_j$ + U(1,20)
w_j	U(1,11)	igual a s_j	U(1,9)
s_j	U(1,30) U(15,35)	U(4,10), U(4,14)	U(1,10), U(1,20) e U(5,15) se $Q = 20$ U(1,20), U(1,40) e U(10,30) se $Q = 40$
Q	40	20	[20,40]

Tabela 2: Distribuição de parâmetros para geração de instância em trabalhos anteriores e neste trabalho. U(a, b) significa distribuição uniforme com valor mínimo *a* e máximo *b*. Com relação ao r_j, para o conjunto proposto foram geradas instâncias para cada valor de $R_2 = \{0,1; 0,4; 0,8\}$.

Grupo	$Q = 20$	$Q = 40$
G1	$s \in [1; 10]; R_2 = 0,1$	$s \in [1; 20]; R_2 = 0,1$
G2	$s \in [1; 10]; R_2 = 0,4$	$s \in [1; 20]; R_2 = 0,4$
G3	$s \in [1; 10]; R_2 = 0,8$	$s \in [1; 20]; R_2 = 0,8$
G4	$s \in [5; 15]; R_2 = 0,1$	$s \in [10; 30]; R_2 = 0,1$
G5	$s \in [5; 15]; R_2 = 0,4$	$s \in [10; 30]; R_2 = 0,4$
G6	$s \in [5; 15]; R_2 = 0,8$	$s \in [10; 30]; R_2 = 0,8$
G7	$s \in [1; 20]; R_2 = 0,1$	$s \in [1; 40]; R_2 = 0,1$
G8	$s \in [1; 20]; R_2 = 0,4$	$s \in [1; 40]; R_2 = 0,4$
G9	$s \in [1; 20]; R_2 = 0,8$	$s \in [1; 40]; R_2 = 0,8$

Tabela 3: Correspondência entre grupos de instâncias e configurações de parâmetro.

do *g++* v5.4.0 com a opção de compilação (otimização) -O3 em uma máquina Intel Core i7 com 3,4 GHz e 16 GB de memória RAM e com sistema operacional Ubuntu 16.04.3 LTS. Apenas uma *thread* foi utilizada para a execução do método em cada instância.

3.3. Comparação com os limitantes conhecidos

Um conjunto de limitantes para instâncias de até 50 tarefas foram disponibilizados por um algoritmo exato (limitado por tempo) que está em desenvolvimento [Pessoa, 2018]. Todos os 266 ótimos comprovados foram encontrados pelo ILS-Batch em todas as 10 execuções do método para cada instância correspondente.

Na Tabela 4 são reportados os *gaps* para os melhores limitantes conhecidos e para a melhor solução viável conhecida (denotada por *BKS* de *best known solution*). Esses valores dizem respeito ao *gap* médio obtido por 10 execuções independentes para as 5 instâncias de cada conjunto. Por exemplo, a coluna *Avg* de Gap_{LB} para 10 tarefas, $Q = 20$ e G1 significa a média entre os *gaps* médios das 10 execuções independentes para as 5 instâncias desse conjunto, e a coluna *Min* significa a menor média de *gap* para os limitantes das 5 instâncias obtida por uma das 10 execuções.

Para instâncias de 10 tarefas, todos os ótimos foram encontrados, uma vez que o *gap* reportado para os limitantes é de 0%. O mesmo ocorre para instâncias com 15 tarefas, com exceção do grupo G4 para $Q = 20$. Nota-se pelos *gaps* reportados que o ILS-Batch é um método bastante robusto. Até o grupo G4 de 20 tarefas nenhum desvio padrão pode ser observado, uma vez que todas as métricas são iguais. O crescimento do *gap* para os limitantes a partir dos conjuntos com 20 tarefas não pode ser atribuído exclusivamente ao ILS-Batch, principalmente devido a tendência de que esses limitantes se tornem mais fracos com o crescimento das instâncias. Deve-se destacar que o tempo de execução do método para instâncias de 50 tarefas foi menor do que 10 segundos, para ambas as capacidades.

4. Considerações finais e trabalhos futuros

Neste artigo foi proposto uma simples e efetiva meta-heurística para resolução do problema de escalonamento de tarefas em uma máquina de processamento em lote com capacidade limitada, cujo objetivo é minimizar o total de penalidades por atraso de tarefas — ou simplesmente

https://proceedings.science/p/85378?lang=pt-br

Tabela 4: Resultados computacionais para instâncias com $Q = 20$ e $Q = 40$.

n	Grupo	$Q = 20$							$Q = 40$						
		Gap_{LB} (%)			Gap_{BKS} (%)			Tempo	Gap_{LB} (%)			Gap_{BKS} (%)			Tempo
		Avg	Min	Max	Avg	Min	Max	(*s*)	Avg	Min	Max	Avg	Min	Max	(*s*)
10	G1	0,00	0,00	0,00	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,00	0,00	0,00	0,02
	G2	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,02
	G3	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,03
	G4	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,02
	G5	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,03
	G6	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,03
	G7	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,02
	G8	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,02
	G9	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,03
15	G1	0,00	0,00	0,00	0,00	0,00	0,00	0,05	0,00	0,00	0,00	0,00	0,00	0,00	0,06
	G2	0,00	0,00	0,00	0,00	0,00	0,00	0,07	0,00	0,00	0,00	0,00	0,00	0,00	0,06
	G3	0,00	0,00	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,09
	G4	0,69	0,69	0,69	0,00	0,00	0,00	0,07	0,00	0,00	0,00	0,00	0,00	0,00	0,07
	G5	0,00	0,00	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,07
	G6	0,00	0,00	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,09
	G7	0,00	0,00	0,00	0,00	0,00	0,00	0,07	0,00	0,00	0,00	0,00	0,00	0,00	0,06
	G8	0,00	0,00	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,07
	G9	0,00	0,00	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,09
20	G1	4,47	4,47	4,47	0,00	0,00	0,00	0,12	5,98	5,98	5,98	0,00	0,00	0,00	0,14
	G2	5,51	5,51	5,51	0,00	0,00	0,00	0,15	0,00	0,00	0,00	0,00	0,00	0,00	0,19
	G3	0,00	0,00	0,00	0,00	0,00	0,00	0,21	0,00	0,00	0,00	0,00	0,00	0,00	0,26
	G4	17,30	17,30	17,30	0,00	0,00	0,00	0,18	11,93	11,93	11,93	0,00	0,00	0,00	0,18
	G5	13,32	13,02	13,63	0,42	0,00	0,84	0,19	3,93	3,93	3,93	0,00	0,00	0,00	0,22
	G6	1,45	1,45	1,45	0,00	0,00	0,00	0,25	0,00	0,00	0,00	0,00	0,00	0,00	0,27
	G7	4,54	4,54	4,54	0,00	0,00	0,00	0,17	6,76	6,76	6,76	0,00	0,00	0,00	0,19
	G8	4,76	4,76	4,76	0,00	0,00	0,00	0,21	3,14	3,14	3,14	0,00	0,00	0,00	0,22
	G9	0,00	0,00	0,00	0,00	0,00	0,00	0,24	0,00	0,00	0,00	0,00	0,00	0,00	0,26
25	G1	18,64	18,50	18,71	0,19	0,00	0,28	0,28	19,45	19,45	19,45	0,00	0,00	0,00	0,34
	G2	18,82	18,82	18,82	0,00	0,00	0,00	0,35	21,93	21,93	21,93	0,00	0,00	0,00	0,34
	G3	5,81	5,81	5,81	0,00	0,00	0,00	0,48	20,89	20,89	20,89	0,00	0,00	0,00	0,48
	G4	20,46	20,46	20,46	0,00	0,00	0,00	0,41	15,07	15,07	15,07	0,00	0,00	0,00	0,41
	G5	17,55	17,54	17,64	0,01	0,00	0,14	0,55	15,74	15,74	15,74	0,00	0,00	0,00	0,41
	G6	14,11	14,11	14,11	0,00	0,00	0,00	0,57	5,76	5,76	5,76	0,00	0,00	0,00	0,61
	G7	13,41	13,39	13,44	0,03	0,00	0,06	0,39	16,95	16,93	17,03	0,03	0,00	0,13	0,39
	G8	13,00	13,00	13,00	0,00	0,00	0,00	0,52	13,65	13,65	13,65	0,00	0,00	0,00	0,58
	G9	12,28	12,28	12,28	0,00	0,00	0,00	0,58	0,00	0,00	0,00	0,00	0,00	0,00	0,59
30	G1	23,93	23,92	23,94	0,01	0,00	0,03	0,54	21,47	21,45	21,49	0,03	0,00	0,04	0,53
	G2	23,99	23,93	24,07	0,07	0,00	0,17	0,62	30,32	30,29	30,44	0,05	0,00	0,24	0,81
	G3	23,80	23,80	23,80	0,00	0,00	0,00	0,78	32,80	32,80	32,83	0,01	0,00	0,04	1,01
	G4	14,41	14,41	14,41	0,00	0,00	0,00	0,85	14,16	14,16	14,16	0,00	0,00	0,00	0,69
	G5	20,05	20,05	20,05	0,00	0,00	0,00	0,96	15,62	15,62	15,62	0,00	0,00	0,00	1,10
	G6	26,25	26,25	26,25	0,00	0,00	0,00	1,32	13,85	13,85	13,85	0,00	0,00	0,00	1,31
	G7	21,09	21,09	21,09	0,00	0,00	0,00	0,73	22,69	22,69	22,69	0,00	0,00	0,00	0,72
	G8	16,28	16,28	16,28	0,00	0,00	0,00	1,09	22,21	22,19	22,26	0,02	0,00	0,09	0,90
	G9	9,54	9,54	9,56	0,00	0,00	0,02	1,11	18,21	18,16	18,28	0,06	0,00	0,15	1,12
35	G1	22,97	22,97	22,97	0,00	0,00	0,00	0,80	22,62	22,62	22,67	0,01	0,00	0,07	0,88
	G2	32,02	31,97	32,25	0,08	0,00	0,39	1,14	23,82	23,82	23,82	0,00	0,00	0,00	1,04
	G3	27,15	27,10	27,53	0,06	0,00	0,61	1,49	30,07	30,07	30,07	0,00	0,00	0,00	1,48
	G4	15,90	15,90	15,90	0,00	0,00	0,00	1,34	18,33	18,25	18,45	0,09	0,00	0,24	1,67
	G5	23,16	23,16	23,16	0,00	0,00	0,00	1,93	24,50	24,41	24,65	0,13	0,00	0,34	1,90
	G6	24,55	24,55	24,55	0,00	0,00	0,00	2,23	21,03	21,03	21,03	0,00	0,00	0,00	2,32
	G7	15,59	15,59	15,59	0,00	0,00	0,00	1,28	13,66	13,66	13,66	0,00	0,00	0,00	1,32
	G8	20,68	20,68	20,68	0,00	0,00	0,00	1,74	19,86	13,17	20,62	0,00	0,00	0,01	1,87
	G9	21,43	21,43	21,43	0,00	0,00	0,00	2,17	16,52	16,52	16,52	0,00	0,00	0,00	2,28
50	G1	22,40	22,26	22,72	0,36	0,18	0,75	3,37	20,42	20,35	20,52	0,09	0,00	0,21	4,73
	G2	29,94	29,84	30,15	0,18	0,04	0,49	4,22	26,42	26,34	26,66	0,11	0,00	0,42	5,38
	G3	36,65	36,50	36,94	0,24	0,00	0,71	5,60	35,51	32,94	39,36	0,05	0,00	0,11	5,48
	G4	16,24	16,24	16,24	0,00	0,00	0,00	6,01	17,96	17,92	18,08	0,05	0,00	0,20	5,75
	G5	24,84	24,82	24,88	0,03	0,00	0,09	7,23	23,36	23,34	23,54	0,03	0,00	0,27	7,21
	G6	36,75	36,75	36,75	0,00	0,00	0,00	8,52	29,40	29,40	29,40	0,00	0,00	0,01	9,10
	G7	16,64	16,58	16,66	0,07	0,00	0,09	6,19	16,61	16,58	16,76	0,04	0,00	0,23	5,94
	G8	25,62	25,58	25,70	0,06	0,00	0,17	7,29	24,41	24,33	24,53	0,09	0,00	0,25	6,65
	G9	32,37	32,24	32,50	0,21	0,00	0,42	9,75	39,50	39,47	39,60	0,06	0,00	0,23	8,08

https://proceedings.science/p/85378?lang=pt-br

$1|batch, r_j, s_j, compt| \sum w_j T_j$. Foram utilizadas cinco estruturas de vizinhança e, para duas delas, uma discussão detalhada sobre a avaliação do movimento foi realizada. Além disso, um novo conjunto de instâncias para o problema foi proposto para gerar instâncias mais difíceis que os *benchmarks* existentes na literatura.

Os resultados computacionais mostraram que o método proposto foi capaz de encontrar todos os 266 ótimos comprovados de forma robusta. Além disso, para todos os grupos de instâncias, o método foi executado com tempo médio abaixo dos 10 segundos.

Como trabalhos futuros, pode-se estudar estratégias de limitação como filtros de movimentos no *best improvement* que sejam capazes de reduzir consideravelmente a complexidade do procedimento sem perda abrupta na sua capacidade de convergência. Além disso, o desenvolvimento de algoritmos híbridos que combinem o ILS-Batch com métodos exatos, de forma similar a abordagem proposta em Subramanian et al. [2013], pode ser abordado.

Referências

Balasubramanian, H., Mönch, L., Fowler, J., e Pfund, M. (2004). Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *International Journal of Production Research*, 42(8):1621–1638.

Chou, F.-D. e Wang, H.-M. (2008). Scheduling for a single semiconductor batch-processing machine to minimize total weighted tardiness. *Journal of the Chinese Institute of Industrial Engineers*, 25(2):136–147.

Hansen, P., Mladenović, N., Brimberg, J., e Pérez, J. A. M. (2010). Variable neighborhood search. In Gendreau, M. e Potvin, J.-Y., editors, *Handbook of Metaheuristics*, p. 61–86. Springer US, Boston, MA. ISBN 978-1-4419-1665-5.

Kramer, A. e Subramanian, A. (2017). A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems. *Journal of Scheduling*. ISSN 1099-1425.

Lee, C.-Y. (1999). Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research*, 37(1):219–236.

Lee, C.-Y., Uzsoy, R., e Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775.

Lourenço, H. R., Martin, O. C., e Stützle, T. (2010). Iterated local search: Framework and applications. In Gendreau, M. e Potvin, J.-Y., editors, *Handbook of Metaheuristics*, p. 363–397. Springer US, Boston, MA. ISBN 978-1-4419-1665-5.

Martins, I. C., Pinheiro, R. G., Protti, F., e Ochi, L. S. (2015). A hybrid iterated local search and variable neighborhood descent heuristic applied to the cell formation problem. *Expert Systems with Applications*, 42(22):8947–8955. ISSN 0957-4174.

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):990–1001. ISSN 1433-3015.

Mathirajan, M., Bhargav, V., e Ramachandran, V. (2010). Minimizing total weighted tardiness on a batch-processing machine with non-agreeable release times and due dates. *The International Journal of Advanced Manufacturing Technology*, 48(9):1133–1148. ISSN 1433-3015.

Parsa, N. R., Karimi, B., e Husseini, S. M. (2016). Minimizing total flow time on a batch processing machine using a hybrid max–min ant system. *Computers & Industrial Engineering*, 99:372 – 381. ISSN 0360-8352.

https://proceedings.science/p/85378?lang=pt-br

Penna, P. H. V., Subramanian, A., e Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19(2):201–232.

Perez, I. C., Fowler, J. W., e Carlyle, W. (2005). Minimizing total weighted tardiness on a single batch process machine with incompatible job families. *Computers & Operations Research*, 32 (2):327 – 341. ISSN 0305-0548.

Pessoa, N. V., A. (2018). Comunicação pessoal.

Peter, B., Andrei, G., Han, H., Y., K. M., N., P. C., Thomas, T., e van de Velde Steef L. Scheduling a batching machine. *Journal of Scheduling*, 1(1):31–54.

Subramanian, A., Uchoa, E., e Ochi, L. S. (2013). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40:2519–2531.

Subramanian, A. e Farias, K. (2017). Efficient local search limitation strategy for single machine total weighted tardiness scheduling with sequence-dependent setup times. *Computers & Operations Research*, 79:190–206. ISSN 0305-0548.

Talbi, E.-G. (2013). *Hybrid Metaheuristics*. Springer Publishing Company, Incorporated. ISBN 364242905X, 9783642429057.

Vélez-Gallego, M. C., Maya, J., e Damodaran, P. (2011). Minimizing total weighted tardiness on a batch-processing machine with non-identical job sizes and non-zero ready times. In *IIE Annual Conference. Proceedings*, p. 1. Institute of Industrial and Systems Engineers (IISE).

Vepsalainen, A. P. J. (1987). Priority rules for job shops with weighted tardiness costs. *Manage. Sci.*, 33(8):1035–1047. ISSN 0025-1909.

Wang, H.-M. (2011). Solving single batch-processing machine problems using an iterated heuristic. *International Journal of Production Research*, 49(14):4245–4261.

Xu, R., Chen, H., e Li, X. (2012). Makespan minimization on single batch-processing machine via ant colony optimization. *Computers & Operations Research*, 39(3):582 – 593. ISSN 0305-0548.

Zinouri, N., Muskeyvalley, R., Damodaran, P., e Ghrayeb, O. (2012). Scheduling a batch processing machine to minimize total weighted tardiness. In *62nd IIE Annual Conference and Expo 2012*, p. 692–701. Institute of Industrial and Systems Engineers (IISE), Institute of Industrial Engineers.

https://proceedings.science/p/85378?lang=pt-br