LIV Simpósio Brasileiro de Pesquisa Operacional
Juiz de Fora, MG - 8 a 11 de novembro de 2022

Uma heurística para o problema de *flow shop* com bloqueio e minimização do *makespan*

Ewerton Victor Paredes da Penha Teixeira
UFPB - Programa de Pós-Graduação em Engenharia de Produção e Sistemas
Cidade Universitária, s/n - Castelo Branco III, João Pessoa - PB, 58059-900
ewertonvppt@gmail.com
Anand Subramanian
UFPB - Departamento de Sistemas de Computação
Rua dos Escoteiros – Mangabeira, 58058-600, João Pessoa, PB
anand@ci.ufpb.br
Hugo Harry Kramer
UFPB - Departamento de Engenharia de Produção
Cidade Universitária, s/n - Castelo Branco III, João Pessoa - PB, 58059-900
hugo.kramer@academico.ufpb.br

RESUMO

Este artigo aborda o problema de *flow shop* com bloqueio para minimização do *makespan*. Nesse problema, n tarefas precisam ser escalonadas em um ambiente com m máquinas ordenadas em série, em que todas as tarefas devem seguir a mesma ordem de processamento e o estoque intermediário entre duas máquinas consecutivas é nulo. É proposto um algoritmo baseado na meta-heurística *iterated local search* que inclui métodos de aceleração da busca local para resolver o problema. Experimentos computacionais foram realizados em 120 instâncias de *benchmark*. O método proposto obteve soluções competitivas, sendo 26 delas melhores do que as encontradas na literatura.

PALAVRAS CHAVE. Flow shop; Bloqueio; Iterated Local Search.

Tópicos: Metaheurísticas; PO na Administração e Gestão da Produção.

ABSTRACT

This paper addresses the blocking flow shop scheduling problem with makespan minimization. In this problem, n jobs must be scheduled in an environment with m ordered machines ordered in series, where each job follows the same processing order. There is no intermediate buffers between two consecutive machines. In order to solve the problem, we propose an algorithm based on the iterated local search metaheuristic whose local search includes acceleration methods. Computational experiments were performed on 120 benchmark instances. The proposed method obtained competitive solutions, in which 26 of them were better than the ones reported in the literature.

KEY WORDS. Flow shop; Blocking; Iterated Local Search.

Paper topics: Metaheuristics; OR in Administration and Production Management.

https://proceedings.science/p/157562?lang=pt-br

1 Introdução

O problema de sequenciamento de tarefas consiste em determinar uma ordem de processamento de n tarefas que devem ser processadas em m máquinas com o intuito de otimizar determinado objetivo. Nesse sentido, o presente trabalho estuda o problema de minimização do *makespan* em um ambiente de *flow shop* com bloqueio. Tal ambiente caracteriza-se por apresentar um conjunto de m máquinas organizadas em série, em que todas as n tarefas devem ser processadas na mesma sequência, conforme a ordem das máquinas. O problema estudado considera que o estoque intermediário entre cada uma das máquinas é nulo, ou seja, caso uma tarefa termine seu processamento em uma máquina i e a máquina $i+1$ não estiver disponível para recebê-la, ocorre um bloqueio na máquina i, que não pode iniciar o processamento da próxima tarefa na sequência. Tal problema é NP-Difícil para $m \geq 3$ [Hall e Sriskandarajah, 1996].

A partir do exposto, o problema em estudo aparece como um problema prático e que merece atenção para garantir que os recursos sejam gerenciados da maneira mais adequada. De acordo com Hall e Sriskandarajah [1996], uma das causas de bloqueio é a eventual impossibilidade de utilização de estoque intermediário entre as máquinas. Miyata e Nagano [2019] apontam que outra causa reside na própria tecnologia de produção, como temperatura e características dos materiais, que requerem que a tarefa finalizada em uma máquina i permaneça neste local até que a seguinte possa recebê-la de modo a evitar deterioração na tarefa, retrabalho e custos adicionais. Liu e Kozan [2009] e Chen et al. [2014] estudaram o problema na indústria química e de eletrônicos, respectivamente, e mostraram que o bloqueio é algo inerente às características dos processos. Outros sistemas de produção também podem ser modelados utilizando o problema de *flow shop* com bloqueio, como indústrias farmacêuticas [Hall e Sriskandarajah, 1996], linhas de produção *just-in-time* [Prasad et al., 2006] e células robóticas [Ribas et al., 2015].

Dessa forma, é proposto um algoritmo baseado na meta-heurística *Iterated Local Search* [Ramalhinho Lourenço et al., 2019] cuja etapa de busca local inclui métodos de aceleração que evitam a avaliação de movimentos que não levam a uma melhora da solução. O método proposto foi avaliado utilizando instâncias de *benchmark* e os resultados obtidos foram comparados com limites superiores já conhecidos na literatura.

O restante do trabalho encontra-se organizado como segue. Uma breve revisão da literatura é apresentada na Seção 2. A Seção 3 descreve o problema de maneira formal. O algoritmo proposto é descrito na Seção 4. A Seção 5 contém os experimentos computacionais, e as conclusões e sugestões de trabalhos futuros são expostos na Seção 6.

2 Revisão da Literatura

Dada a sua importância prática, variantes do problema de *flow shop* com bloqueio são estudadas desde o trabalho de Levner [1969]. Heurísticas construtivas, métodos exatos e algoritmos baseados em meta-heurísticas já foram propostos para o problema em estudo. Entre as heurísticas construtivas encontradas na literatura, destacam-se heurísticas como a *Profile Fitting* [McCormick et al., 1989], MinMax [Ronconi, 2004], heurísticas de Pan e Wang [2012] e o algoritmo NEH [Nawaz et al., 1983].

Algoritmos baseados em meta-heurísticas compõem grande parte dos trabalhos propostos para o problema. Entre os diversos tipos de meta-heurísticas encontrados na literatura, os seguintes já foram propostos para resolver problemas de *flow shop* com bloqueio: *Iterated Greedy Algorithm*

https://proceedings.science/p/157562?lang=pt-br

[Ribas et al., 2011; Ding et al., 2016], Algoritmo Memético [Pan et al., 2012], Algoritmo Genético [Caraffa et al., 2001], Algoritmos de três fases [Wang e Tang, 2012], *Simulated Annealing* [Zhang et al., 2015], *Differential Evolution Algorithm* [Wang et al., 2010; Davendra et al., 2012], entre outros.

Todavia, o estado da arte é o algoritmo prosposto por Shao et al. [2019]. Os autores desenvolveram um algoritmo denominado *Discrete Invasive Weed Optimization* (DIWO) que faz uso de uma heurística construtiva denominada PF-NEH [Pan e Wang, 2012] para construir uma solução inicial de alta qualidade. Em seguida, o restante da população é construída de forma aleatória. Para manter a capacidade de busca e eficiência, uma dispersão espacial baseada em inserção aleatória é apresentada por meio da distribuição normal com o intuito gerar novos indivíduos da população. Além disso, uma busca local que faz uso de uma solução de referência é incorporada para aumentar ainda mais a capacidade de exploração e Um método de exclusão de indivíduos é utilizado para determinar uma população de descendentes com boa qualidade e diversidade. Por fim, alguns métodos exatos para resolução do problema também já foram propostos: *branch-and-bound* [Levner, 1969; Ronconi, 2005; Toumi et al., 2017], modelos de programação inteira mista [Pitty e Karimi, 2008] e *bounded dynamic programming* [Bautista et al., 2012; Ozolins, 2019].

3 Descrição do Problema

Seja n o total de tarefas que devem ser processadas em um conjunto de máquinas ordenadas $M = (M_1, M_2, \ldots, M_i, \ldots, M_m)$, o objetivo do problema consiste em achar uma permutação de tarefas $\pi = (\pi_1, \pi_2, \ldots, \pi_k, \ldots, \pi_n)$, em que π_k denota a k-ésima tarefa da sequência, que minimize o *makespan* (C_{max}). Considere que $p_{\pi_k,i}$ consiste no tempo de processamento de π_k em M_i. Seja $D_{\pi_k,i}$ o momento em que π_k deixa M_i. Note que $D_{\pi_k,i} \geq C_{\pi_k,i}$, em que $C_{\pi_k,i}$ denota o tempo de conclusão do processamento de π_k em M_i. O caso em que $D_{\pi_k,i} = C_{\pi_k,i}$ ocorre quando não há um bloqueio entre M_i e M_{i+1}. O *makespan* é dado por $C_{max} = D_{\pi_n,m}$. Sendo assim, Ronconi [2004] mostra que o *makespan* de uma permutação π pode ser calculado conforme segue:

$$D_{\pi_1,i} = \sum_{l=1}^{i} p_{\pi_1,l} \qquad i = 1, \ldots, m \qquad (1)$$

$$D_{\pi_k,i} = max(D_{\pi_k,i-1} + p_{\pi_k,i}, D_{\pi_{k-1},i+1}) \qquad k = 2, \ldots, n; i = 1, \ldots, m-1 \qquad (2)$$

$$D_{\pi_k,m} = D_{\pi_k,m-1} + p_{\pi_k,m} \qquad k = 2, \ldots, n; \qquad (3)$$

A Equação (1) calcula os tempos de saída da primeira tarefa da permutação π de cada uma das máquinas, em que a tarefa não encontra bloqueio. Já a Equação (2) calcula os tempos de saída das demais tarefas de cada uma das máquinas, com exceção da máquina m. Neste caso, verifica-se se a tarefa poderá sair da máquina i ao terminar seu processamento ou se deverá esperar a tarefa $k-1$ acabar de ser processada na máquina $i+1$. Por fim, os tempos de saída das tarefas da máquina m são dados pela Equação (3).

Outra forma de calcular o *makespan* é através da representação em grafo do problema, como o definido por Grabowski e Pempera [2007]. Seja $G(\pi) = (N, F^+ \cup F^0 \cup F^-)$ um grafo

https://proceedings.science/p/157562?lang=pt-br

com um conjunto de nós $N = n \times m$ e um conjunto de arcos $F^+ \cup F^0 \cup F^-$. Cada nó $(k, i) \in G$ tem peso igual a $p_{\pi_k,i}$ e o conjunto de arcos é definido como segue:

$$F^+ = \bigcup_{k=1}^{n} \bigcup_{i=2}^{m} \{((k, i-1), (k, i))\} \tag{4}$$

$$F^0 = \bigcup_{k=2}^{n} \bigcup_{i=1}^{m} \{((k-1, i), (k, i))\} \tag{5}$$

$$F^- = \bigcup_{k=2}^{n} \bigcup_{i=2}^{m} \{((k-1, i), (k, i-1))\} \tag{6}$$

As Equações (4)–(6) se referem a transições entre máquinas, a transições entre tarefas e às restrições de bloqueio, respectivamente. Cada arco $\{((k-1, i), (k, i-1))\} \in F^-$ garante o bloqueio da tarefa π_k na máquina i se a máquina $i+1$ não estiver liberada. Esses arcos têm pesos iguais a $-p_{\pi_{k-1},i}$, enquanto os arcos pertencentes a F^+ e F^0 têm pesos iguais a zero. O *makespan* de uma permutação π é dado pelo caminho mais longo (crítico) entre os nós $(1, 1)$ e (n, m) do grafo $G(\pi)$.

4 Algoritmo Proposto

O método utilizado para resolução do problema é baseado na meta-heurística *Iterated Local Search* (ILS), com múltiplas inicializações, aqui denominado ILS_{BFSP}. O algoritmo possui dois parâmetros principais, que são o número máximo de iterações da busca local (I_{ILS}) e o parâmetro da perturbação (d). Após inicializar uma solução vazia, inicia-se o laço de reinícios do algoritmo, em que, a cada uma das iterações, uma nova solução inicial é gerada. Após esta etapa, tem-se o laço principal do algoritmo: caso a solução obtida pela busca local melhore a função objetivo, a nova solução é armazenada como a melhor encontrada pela busca local e o contador de iterações é reinicializado com zero. Em seguida, a melhor solução armazenada é perturbada e atribuída à solução corrente. O processo continua até que I_{ILS} iterações da busca local sejam executadas. Caso a melhor solução ao fim da busca local seja melhor que a melhor solução conhecida, esta é atualizada e inicia-se uma nova inicialização do algoritmo. Ao atingir o critério de parada, a melhor solução é retornada.

4.1 Geração da Solução Inicial

Conforme mencionado, uma solução inicial é obtida em cada uma das reinicializações. Nesse sentido, é importante que, a cada reinicialização do algoritmo, uma nova solução diferente das anteriores seja obtida. O intuito é fazer com que haja maior diversidade de soluções para que a busca local seja aplicada. Dessa forma, foi escolhida como heurística construtiva o algoritmo NEH. O algoritmo original assume que uma tarefa com o maior tempo total de processamento deve iniciar o sequenciamento e a insere em uma permutação parcial π'. Em seguida, para cada uma das $n-1$ tarefas restantes, o algoritmo seleciona a posição relativa de π' que proporciona o melhor *makespan* e insere a tarefa em tal posição. No final das $n-1$ iterações, é formada uma permutação π. Como pode ser observado, o algoritmo NEH é determinístico. Desta forma, uma mesma permutação π é gerada sempre que uma determinada tarefa k é escolhida para iniciar o método. Assim, para que sejam geradas soluções diferentes a cada iteração do ILS, optou-se que a primeira tarefa k a ser

inserida na permutação π seja escolhida de forma completamente aleatória e, em seguida, as demais tarefas são inseridas normalmente.

4.2 Busca Local

A fase de busca local utiliza um conjunto N de estruturas de vizinhança que são exploradas de forma aleatória. Quando uma estrutura de vizinhança não melhora a solução corrente, ela é removida de N. Por outro lado, quando ela consegue obter uma melhora, todas as estruturas são reinseridas em N. O procedimento chega ao fim quando todas as estruturas de vizinhança são exploradas e nenhuma melhora é obtida. Um total de 5 estruturas de vizinhança foram utilizadas:

Block Insertion: Um bloco de x tarefas consecutivas é retirado de sua posição na solução π e é reinserido em uma nova posição, em que $1 \leq x \leq 4$.
Swap: Duas tarefas da solução π são trocadas de posição entre si.

Conforme as Equações (1)-(3), a complexidade computacional para o cálculo do *makespan* de uma permutação π é de $O(nm)$. Desta forma, a complexidade computacional de se explorar as vizinhanças *swap* e *block insertion* é de $O(n^3m)$. Assim, faz-se necessário a construção de métodos que diminuam esta complexidade de modo que a busca local seja acelerada.

Na literatura, algumas propriedades de bloqueio já foram propostas. Wang et al. [2010] propuseram um método para reduzir a complexidade computacional de se explorar uma vizinhança de inserção para $O(n^2m)$ fazendo uso da propriedade de reversibilidade do *flow shop*. Tal propriedade permite que dada uma sequência π', que possui tarefas ordenadas da forma inversa a uma permutação π, o *makespan* $R_{\pi'_n,m}$ desta sequência é igual a $D_{\pi_n,m}$ se ela for processada em um conjunto de máquinas $M = (M_m, M_{m-1}, \ldots, M_1)$. De modo a acelerar a exploração das vizinhanças *swap* e *block insertion*, optou-se por adaptar o método proposto por Wang et al. [2010]. Ele consiste nos seguintes passos, para duas posições j e q de uma vizinhança do tipo *block insertion*, com $q > j$:

1. Calcule os tempos de saída $D_{\pi_k,i}$ e $R_{\pi'_k,i}$, para $k = 1, \ldots, n$ e $i = 1, \ldots, m$.
2. Faça $j = 1$.
3. Faça $\pi'' = (\pi''_1, \pi''_2, \ldots, \pi''_{n-x})$ uma permutação parcial após a remoção de $(\pi_j, \pi_{j+1}, \ldots, \pi_{j+x-1})$ de π.
4. Calcule os tempos de saída $D_{\pi''_k,i}(k = 1, \ldots, n-x, i = 1, \ldots, m)$ e $R_{\pi''_k,i}(k = n-x, \ldots, 1, i = 1, \ldots, m)$
5. Repita os seguintes passos até todas as q possíveis posições, $q \in (1, 2, \ldots, n) \wedge q < j + x$, serem avaliadas:
   (a) Insira o bloco $b = (\pi_j, \pi_{j+1}, \ldots, \pi_{j+x-1})$ na posição q e gere uma permutação π_r.
   (b) Calcule os tempos de saída das tarefas de b a partir de $D_{\pi_{r_{q-1}},i}$, com $i = 1, \ldots, m$.

(c) O *makespan* de π_r é dado por:

$$C_{max} = \max_{i=1}^{m}(D_{\pi_{r_{q+x-1}},i} + R_{\pi''_{q+x-1},i}) \tag{7}$$

6. Faça $j = j + 1$. Se $j > n - x$, pare. Caso contrário, volte para o passo 4.

Observe que há um total de n iterações para os passos 4 e 5, que podem ser executados em $O(nm)$. Assim, para executar toda vizinhança, a complexidade computacional é de $O(n^2m)$. Além disso, observe que o item (c) do passo 5 não precisa ser executado até o final. Caso em uma determinada iteração i o valor seja maior ou igual ao *makespan* corrente, não há a necessidade de avaliar as demais iterações.

Para a vizinhança *swap*, um método similar foi desenvolvido. Note que ao trocar duas tarefas j e q de posição, com $q > j$, os tempos de saída $D_{\pi_k,i}$ se mantêm para $k < j$. Desta forma, basta calcular os novos tempos de saída $D_{\pi_k,i}, j \leq k \leq q$, e aplicar a propriedade de reversibilidade conforme a Equação (7).

Além dos métodos de aceleração propostos, foram implementados as propriedades de bloqueio propostas por Grabowski e Pempera [2007] e o limitante inferior introduzido por Ding et al. [2016] para vizinhança *block insertion*, no caso em que $x = 1$. Tais propriedades são baseadas a partir do conceito da representação do problema em grafo, conforme ilustrado na Seção 2 e apresentam condições suficientes para se obter uma permutação π' a partir de π de tal forma que $C_{max}(\pi') < C_{max}(\pi)$. Ainda, foram desenvolvidos limitantes inferiores para a vizinhança *swap*.

Com o intuito de elucidar tais limitantes, faz-se necessário explicar os conceitos de tipos de blocos, formulados por Grabowski e Pempera [2007]. Em um caminho definido no grafo $G(\pi)$, um bloco é formado quando nós são ligados, de forma consecutiva, por um único tipo de arco. Como há três tipos de arcos diferentes, existem três tipos de blocos diferentes, que são: *normal block* (GH), formado por arcos F^+; *anti-block* (SH), formado por arcos F^- e *T-sequence* (T), formado por arcos F^0. As máquinas nas quais um bloco q inicia e termina são denotadas por s_q e e_q, respectivamente. Assim, os limitantes inferiores para uma vizinhança *swap* são dados por:

Limitante Inferior 1 — GH: Seja π' uma permutação obtida por um movimento de troca entre duas tarefas j e l, onde $j < l$ em que j pertence a um *normal block* t e l a um tipo de bloco q. Então, $C_{max}(\pi') \geq C_{max}(\pi) + \delta_{jl}(\pi)$, onde:

$$\delta_{jl}(\pi) = \begin{cases} p_{\pi_l s_t} - p_{\pi_j s_t} + p_{\pi_j s_q} - p_{\pi_l s_q}, & q \in GH \\ p_{\pi_l s_t} - p_{\pi_j s_t}, & q \in ST \\ p_{\pi_l s_t} - p_{\pi_j s_t} + \sum_{i=s_q}^{e_q} p_{\pi_j i} - p_{\pi_l i} & q \in T \end{cases} \tag{8}$$

Limitante Inferior 2 — ST: Seja π' uma permutação obtida por um movimento de troca entre duas tarefas j e l, onde $j < l$ em que j pertence a um *anti-block* t e l a um tipo de bloco q. Então, $C_{max}(\pi') \geq C_{max}(\pi) + \delta_{jl}(\pi)$, onde:

$$\delta_{jl}(\pi) = \begin{cases} p_{\pi_j s_q} - p_{\pi_l s_q}, & q \in GH \\ 0, & q \in ST \\ \sum_{i=s_q}^{e_q} p_{\pi_j i} - p_{\pi_l i} & q \in T \end{cases} \quad (9)$$

Limitante Inferior 3 — T: Seja $\pi^{'}$ uma permutação obtida por um movimento de troca entre duas tarefas j e l, onde $j < l$ em que j pertence a uma $T - sequence$ t e l a um tipo de bloco q. Então, $C_{max}(\pi^{'}) \geq C_{max}(\pi) + \delta_{jl}(\pi)$, onde:

$$\delta_{jl}(\pi) = \begin{cases} p_{\pi_l s_q} - p_{\pi_j s_q} + \sum_{i=s_t}^{e_t} p_{\pi_l i} - p_{\pi_j i}, & q \in GH \\ \sum_{i=s_t}^{e_t} p_{\pi_l i} - p_{\pi_j i}, & q \in ST \\ \sum_{i=s_t}^{e_t} p_{\pi_l i} - p_{\pi_j i} + \sum_{i=s_q}^{e_q} p_{\pi_j i} - p_{\pi_l i} & q \in T \end{cases} \quad (10)$$

Tais limitantes podem ser validados a partir da noção do caminho crítico relacionado ao grafo $G(\pi)$ [Ding et al., 2016]. Os autores mostram que o comprimento de tal caminho é dado por $C_{max}(\pi) = \sum_{n_i \in V_{GH}} p_{n_i} + \sum_{n_i \in V_T} p_{n_i}$, onde n_i é um nó pertencente ao caminho crítico, V_{GH} é o conjunto de nós que compõem os *normal-blocks* do caminho, V_T o conjunto de nós que compõem as *T-sequences* e p_{n_i} o peso relacionado ao nó n_i. No caso de uma permutação π' obtida a partir da troca de duas tarefas j, pertecente a um bloco $t \in GH$, e l, que está em um bloco $q \in GH$, o *makespan* é dado por $C_{max}(\pi') = \sum_{n_i \in V'_{GH}} p_{n_i} + \sum_{n_i \in V_T} p_{n_i}$, onde $V'_{GH} = V_{GH} + \{(\pi_l, s_t)\} + \{(\pi_j, s_q)\} - \{(\pi_j, s_t)\} - \{(\pi_l, s_q)\}$. Dessa forma, $C_{max}(\pi') = C_{max}(\pi) + p_{\pi_l s_t} - p_{\pi_j s_t} + p_{\pi_j s_q} - p_{\pi_l s_q}$. Os demais casos podem ser mostrados de forma similar.

Em suma, nota-se que um movimento de troca entre duas tarefas j e l só poderá resultar em uma permutação $\pi^{'}$ com $C_{max}(\pi^{'}) < C_{max}(\pi)$ se, e somente se, $\delta_{jl}(\pi) < 0$. O cálculo de $\delta_{jl}(\pi)$ pode ser feito em $O(1)$, reduzindo o tempo computacional de exploração da vizinhança *swap* em sua totalidade. Por fim, para mapear o caminho crítico de uma permutação π e os tipos de bloco em que cada tarefa está inserida, foi utilizado o método de Ding et al. [2016].

4.3 Perturbação

O procedimento de perturbação utilizado consiste em destruir e reconstruir uma permutação π. Após a remoção de d tarefas, escolhidas aleatoriamente, de π, forma-se uma permutação parcial π_D e uma lista (LC) com as tarefas removidas. Enquanto LC não estiver vazia, é selecionada uma tarefa $j \in LC$ de forma aleatória. Em seguida, utiliza-se a fase de inserção do algoritmo NEH para reinserir a tareja j em π_D na posição que resulta no menor *makespan*. Ao final, retorna-se uma permutação π completa.

5 Experimentos Computacionais

Os experimentos computacionais foram realizados nas instâncias propostas por Taillard [1993]. Elas possuem três diferentes níveis de máquinas, com $m = \{5,\ 10$ e $20\}$, e cinco diferentes conjuntos de tarefas, com $n = \{20, 50, 100, 200$ e $500\}$. Estas combinações totalizam 120 instâncias. Os resultados obtidos pelo ILS$_{BFSP}$ foram confrontados com os limites superiores apresentados por Shao et al. [2019]. Os autores compararam os resultados obtidos pelo algoritmo

proposto por eles, denominado *Discrete Invasive Weed Optimization* (DIWO), com outros algoritmos já propostos na literatura.

O algoritmo proposto no presente trabalho foi implementado na linguagem de programação C++ e os experimentos foram realizados em uma máquina com processador Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 8 GB de memória RAM e sistema operacional Linux Ubuntu 20.04. A seguinte parametrização foi utilizada: $I_{ILS} = 3 \times n$ e $d = 8$. O critério de parada do algoritmo foi o mesmo adotado por Shao et al. [2019], sendo o limite de tempo de execução $T = 100 \times n \times m$ milissegundos.

Os resultados obtidos são apresentados na Tabela 1. Cada uma das 120 instâncias foi executada 10 vezes e a tabela se encontra organizada da seguinte forma: a primeira coluna representa o nome da instância, a segunda coluna indica a *best known solution* (BKS), a terceira coluna representa o melhor limitante superior obtido pelo DIWO, as duas colunas seguintes indicam o melhor limitante superior e a média das 10 execuções do ILS_{BFSP}. Em seguida, as quatro próximas colunas indicam os *gaps* mínimos e médios quando comparamos o algoritmo proposto com a BKS e o DIWO e, por fim, a última coluna indica o tempo das execuções do ILS_{BFSP} em segundos.

Tabela 1: Comparação dos resultados do ILS_{BFSP} com limites superiores encontrados na literatura

Instância	BKS	DIWO	Melhor ILS_{BFSP}	Média ILS_{BFSP}	$\text{Gap}_{\text{Melhor-BKS}}$	$\text{Gap}_{\text{Melhor-DIWO}}$	$\text{Gap}_{\text{Média-BKS}}$	$\text{Gap}_{\text{Média-DIWO}}$	Tempo (s)
J20M5N1	1374	1374	1374	1374	0,00%	0,00%	0,00%	0,00%	10
J20M5N2	1408	1408	1408	1408	0,00%	0,00%	0,00%	0,00%	10
J20M5N3	1280	1280	1280	1280	0,00%	0,00%	0,00%	0,00%	10
J20M5N4	1448	1448	1448	1448	0,00%	0,00%	0,00%	0,00%	10
J20M5N5	1341	1341	1341	1341	0,00%	0,00%	0,00%	0,00%	10
J20M5N6	1363	1363	1363	1363	0,00%	0,00%	0,00%	0,00%	10
J20M5N7	1381	1381	1381	1381	0,00%	0,00%	0,00%	0,00%	10
J20M5N8	1379	1379	1379	1379	0,00%	0,00%	0,00%	0,00%	10
J20M5N9	1373	1373	1373	1373	0,00%	0,00%	0,00%	0,00%	10
J20M5N10	1283	1283	1283	1283	0,00%	0,00%	0,00%	0,00%	10
J20M10N1	1698	1698	1698	1698	0,00%	0,00%	0,00%	0,00%	20
J20M10N2	1833	1833	1833	1833	0,00%	0,00%	0,00%	0,00%	20
J20M10N3	1659	1659	1659	1659	0,00%	0,00%	0,00%	0,00%	20
J20M10N4	1535	1535	1535	1535	0,00%	0,00%	0,00%	0,00%	20
J20M10N5	1617	1617	1617	1617	0,00%	0,00%	0,00%	0,00%	20
J20M10N6	1590	1590	1590	1590	0,00%	0,00%	0,00%	0,00%	20
J20M10N7	1622	1622	1622	1622	0,00%	0,00%	0,00%	0,00%	20
J20M10N8	1731	1731	1731	1731	0,00%	0,00%	0,00%	0,00%	20
J20M10N9	1747	1747	1747	1747	0,00%	0,00%	0,00%	0,00%	20
J20M10N10	1782	1782	1782	1782	0,00%	0,00%	0,00%	0,00%	20
J20M20N1	2436	2436	2436	2436	0,00%	0,00%	0,00%	0,00%	40
J20M20N2	2234	2234	2234	2234	0,00%	0,00%	0,00%	0,00%	40
J20M20N3	2479	2479	2479	2479	0,00%	0,00%	0,00%	0,00%	40
J20M20N4	2348	2348	2348	2348	0,00%	0,00%	0,00%	0,00%	40
J20M20N5	2435	2435	2435	2435	0,00%	0,00%	0,00%	0,00%	40
J20M20N6	2383	2383	2383	2383	0,00%	0,00%	0,00%	0,00%	40
J20M20N7	2390	2390	2390	2390	0,00%	0,00%	0,00%	0,00%	40
J20M20N8	2328	2328	2328	2328	0,00%	0,00%	0,00%	0,00%	40
J20M20N9	2363	2363	2363	2363	0,00%	0,00%	0,00%	0,00%	40
J20M20N10	2323	2323	2323	2323	0,00%	0,00%	0,00%	0,00%	40
J50M5N1	**2980**	**2983**	**2974**	**2980,6**	**-0,20%**	**-0,30%**	**0,02%**	**0,02%**	**25**
J50M5N2	**3180**	**3180**	**3177**	**3181,1**	**-0,09%**	**-0,09%**	**0,03%**	**0,03%**	**25**
J50M5N3	**2995**	**3000**	**2989**	**2996,6**	**-0,20%**	**-0,37%**	**0,05%**	**0,05%**	**25**
J50M5N4	**3115**	**3115**	**3114**	**3117,2**	**-0,03%**	**-0,03%**	**0,07%**	**0,07%**	**25**
J50M5N5	**3139**	**3145**	**3138**	**3142**	**-0,03%**	**-0,22%**	**0,10%**	**0,10%**	**25**
J50M5N6	3158	3161	3158	3160,4	0,00%	-0,09%	0,08%	0,08%	25
J50M5N7	**3005**	**3008**	**3004**	**3010,1**	**-0,03%**	**-0,13%**	**0,17%**	**0,17%**	**25**
J50M5N8	**3042**	**3044**	**3039**	**3045,5**	**-0,10%**	**-0,16%**	**0,11%**	**0,11%**	**25**
J50M5N9	2889	2891	2889	2892,4	0,00%	-0,07%	0,12%	0,12%	25
J50M5N10	**3097**	**3102**	**3096**	3101,1	**-0,03%**	**-0,19%**	**0,13%**	**0,13%**	**25**
J50M10N1	**3611**	**3614**	**3605**	**3618**	**-0,17%**	**-0,25%**	**0,19%**	**0,19%**	**50**
J50M10N2	3470	3473	3475	3483,7	0,14%	0,06%	0,39%	0,39%	50
J50M10N3	3465	3465	3466	3471,6	0,03%	0,03%	0,19%	0,19%	50
J50M10N4	**3650**	**3654**	**3645**	**3658**	**-0,14%**	**-0,25%**	**0,22%**	**0,22%**	**50**
J50M10N5	3582	3619	3620	3625,2	1,05%	0,03%	1,19%	1,19%	50
J50M10N6	3571	3575	3579	3585,7	0,22%	0,11%	0,41%	0,41%	50
J50M10N7	3667	3669	3669	3678	0,05%	0,00%	0,30%	0,30%	50
J50M10N8	3549	3549	3553	3560,6	0,11%	0,11%	0,33%	0,33%	50

https://proceedings.science/p/157562?lang=pt-br

Tabela 1: Comparação dos resultados do ILS_{BFSP} com limites superiores encontrados na literatura (continuação)

Instância	BKS	DIWO	Melhor ILS_{BFSP}	Média ILS_{BFSP}	$Gap_{Melhor\text{-}BKS}$	$Gap_{Melhor\text{-}DIWO}$	$Gap_{Média\text{-}BKS}$	$Gap_{Média\text{-}DIWO}$	Tempo (s)
J50M10N9	3508	3518	3510	3521	0,06%	-0,23%	0,37%	0,37%	50
J50M10N10	**3608**	**3610**	**3606**	**3615,7**	**-0,06%**	**-0,11%**	**0,21%**	**0,21%**	**50**
J50M20N1	4479	4479	4490	4494,8	0,24%	0,24%	0,35%	0,35%	100
J50M20N2	4262	4262	4262	4279,9	0,00%	0,00%	0,42%	0,42%	100
J50M20N3	4261	4261	4261	4269,2	0,00%	0,00%	0,19%	0,19%	100
J50M20N4	4339	4342	4347	4353,6	0,18%	0,12%	0,34%	0,34%	100
J50M20N5	4249	4250	4257	4264,2	0,19%	0,16%	0,36%	0,36%	100
J50M20N6	4271	4271	4285	4290,8	0,33%	0,33%	0,46%	0,46%	100
J50M20N7	4291	4296	4299	4305,5	0,19%	0,07%	0,34%	0,34%	100
J50M20N8	4298	4298	4311	4314,2	0,30%	0,30%	0,38%	0,38%	100
J50M20N9	4304	4305	4304	4310,1	0,00%	-0,02%	0,14%	0,14%	100
J50M20N10	4398	4398	4398	4406,6	0,00%	0,00%	0,20%	0,20%	100
J100M5N1	**6065**	**6065**	**6028**	**6047,7**	**-0,61%**	**-0,61%**	**-0,29%**	**-0,29%**	**50**
J100M5N2	**5934**	**5934**	**5926**	**5936,8**	**-0,13%**	**-0,13%**	**0,05%**	**0,05%**	**50**
J100M5N3	**5851**	**5876**	**5824**	**5840**	**-0,46%**	**-0,89%**	**-0,19%**	**-0,19%**	**50**
J100M5N4	**5656**	**5684**	**5649**	**5666,1**	**-0,12%**	**-0,62%**	**0,18%**	**0,18%**	**50**
J100M5N5	**5896**	**5896**	**5873**	**5884,1**	**-0,39%**	**-0,39%**	**-0,20%**	**-0,20%**	**50**
J100M5N6	**5755**	**5755**	**5739**	**5759**	**-0,28%**	**-0,28%**	**0,07%**	**0,07%**	**50**
J100M5N7	**5915**	**5915**	**5893**	**5901**	**-0,37%**	**-0,37%**	**-0,24%**	**-0,24%**	**50**
J100M5N8	**5809**	**5825**	**5788**	**5797,6**	**-0,36%**	**-0,64%**	**-0,20%**	**-0,20%**	**50**
J100M5N9	**6027**	**6027**	**6018**	**6040,8**	**-0,15%**	**-0,15%**	**0,23%**	**0,23%**	**50**
J100M5N10	**6059**	**6059**	**6048**	**6059,2**	**-0,18%**	**-0,18%**	**0,00%**	**0,00%**	**50**
J100M10N1	6906	6906	6940	6960	0,49%	0,49%	0,78%	0,78%	100
J100M10N2	6656	6656	6662	6683,3	0,09%	0,09%	0,41%	0,41%	100
J100M10N3	6797	6807	6798	6823,4	0,01%	-0,13%	0,39%	0,39%	100
J100M10N4	7035	7035	7057	7085,2	0,31%	0,31%	0,71%	0,71%	100
J100M10N5	**6728**	**6728**	**6726**	**6751,4**	**-0,03%**	**-0,03%**	**0,35%**	**0,35%**	**100**
J100M10N6	6537	6549	6543	6572,8	0,09%	-0,09%	0,54%	0,54%	100
J100M10N7	**6689**	**6689**	**6683**	**6720,4**	**-0,09%**	**-0,09%**	**0,47%**	**0,47%**	**100**
J100M10N8	6746	6746	6761	6776,2	0,22%	0,22%	0,45%	0,45%	100
J100M10N9	6928	6958	6952	6969,8	0,35%	-0,09%	0,60%	0,60%	100
J100M10N10	**6855**	**6881**	**6829**	**6856,6**	**-0,38%**	**-0,76%**	**0,02%**	**0,02%**	**100**
J100M20N1	7709	7709	7730	7765,7	0,27%	0,27%	0,73%	0,73%	200
J100M20N2	7744	7769	7791	7810,1	0,60%	0,28%	0,85%	0,85%	200
J100M20N3	7723	7725	7774	7801,8	0,66%	0,63%	1,01%	1,01%	200
J100M20N4	7743	7755	7762	7799,9	0,24%	0,09%	0,73%	0,73%	200
J100M20N5	7730	7744	7770	7796,4	0,51%	0,33%	0,85%	0,85%	200
J100M20N6	7779	7785	7779	7829,9	0,00%	-0,08%	0,65%	0,65%	200
J100M20N7	7857	7857	7882	7915,6	0,32%	0,32%	0,74%	0,74%	200
J100M20N8	7898	7902	7912	7951,4	0,18%	0,13%	0,67%	0,67%	200
J100M20N9	7818	7835	7833	7859,5	0,19%	-0,03%	0,53%	0,53%	200
J100M20N10	7842	7842	7888	7905,4	0,58%	0,58%	0,80%	0,80%	200
J200M10N1	13149	13166	13198	13256,6	0,37%	0,24%	0,81%	0,81%	200
J200M10N2	13085	13085	13116	13165,2	0,24%	0,24%	0,61%	0,61%	200
J200M10N3	13183	13183	13212	13260,7	0,22%	0,22%	0,59%	0,59%	200
J200M10N4	13097	13097	13143	13192,1	0,35%	0,35%	0,72%	0,72%	200
J200M10N5	13100	13100	13129	13227,2	0,22%	0,22%	0,96%	0,96%	200
J200M10N6	12883	12883	12900	12973,5	0,13%	0,13%	0,70%	0,70%	200
J200M10N7	**13408**	**13408**	**13400**	**13455,5**	**-0,06%**	**-0,06%**	**0,35%**	**0,35%**	**200**
J200M10N8	13266	13266	13292	13344,2	0,20%	0,20%	0,59%	0,59%	200
J200M10N9	13080	13080	13090	13132,6	0,08%	0,08%	0,40%	0,40%	200
J200M10N10	13193	13193	13205	13252,4	0,09%	0,09%	0,45%	0,45%	200
J200M20N1	14192	14516	14614	14669,7	2,89%	0,67%	3,26%	3,26%	400
J200M20N2	14714	14714	14788	14862,4	0,50%	0,50%	1,00%	1,00%	400
J200M20N3	14831	14831	14953	15010,3	0,82%	0,82%	1,19%	1,19%	400
J200M20N4	14801	14801	14859	14923,7	0,39%	0,39%	0,82%	0,82%	400
J200M20N5	14609	14609	14698	14749,2	0,61%	0,61%	0,95%	0,95%	400
J200M20N6	**14737**	**14737**	**14733**	**14766,7**	**-0,03%**	**-0,03%**	**0,20%**	**0,20%**	**400**
J200M20N7	14770	14770	14860	14902,2	0,61%	0,61%	0,89%	0,89%	400
J200M20N8	14823	14823	14853	14933,1	0,20%	0,20%	0,74%	0,74%	400
J200M20N9	14691	14691	14758	14818,5	0,45%	0,45%	0,86%	0,86%	400
J200M20N10	14711	14711	14805	14869,5	0,63%	0,63%	1,07%	1,07%	400
J500M20N1	35380	35380	35943	36075	1,57%	1,57%	1,93%	1,93%	1000
J500M20N2	35736	35736	36330	36448,9	1,64%	1,64%	1,96%	1,96%	1000
J500M20N3	35406	35406	35985	36076,5	1,61%	1,61%	1,86%	1,86%	1000
J500M20N4	35030	35738	36197	36289,2	3,22%	1,27%	3,47%	3,47%	1000
J500M20N5	35417	35417	35985	36159,2	1,58%	1,58%	2,05%	2,05%	1000
J500M20N6	35740	35740	36345	36424	1,66%	1,66%	1,88%	1,88%	1000
J500M20N7	35299	35299	35877	35964,5	1,61%	1,61%	1,85%	1,85%	1000
J500M20N8	35515	35515	36159	36248,9	1,78%	1,78%	2,02%	2,02%	1000
J500M20N9	35268	35268	35873	36000,8	1,69%	1,69%	2,04%	2,04%	1000
J500M20N10	35609	35609	36181	36289,6	1,58%	1,58%	1,88%	1,88%	1000

Além de comparar os resultados obtidos com os da literatura, foi computada a média

de soluções não avaliadas a cada passagem do algoritmo pela vizinhança *swap*, visando avaliar o limite inferior proposto. Entre as soluções não avaliadas, estão aquelas que foram evitadas pelas propriedades clássicas propostas por Grabowski e Pempera [2007], e as ignoradas pelos limitantes inferiores propostos no presente trabalho. Os resultados estão sumarizados por grupos de instâncias na Tabela 2. Nota-se que os limites propostos conseguem evitar um número maior de soluções por uma margem considerável quando comparados com as propriedades propostas por Grabowski e Pempera [2007]. Isto acontece porque o limite inferior é derivado a partir de tarefas que podem estar localizadas em blocos distintos no grafo $G(\pi)$, enquanto que as demais propriedades utilizam tarefas pertencentes a um mesmo bloco.

Tabela 2: Média de soluções eliminadas por passagem pela vizinhança *swap*

Instância	Propriedades clássicas	Limite inferior proposto	%Classica	%Proposto
20x5	7,62	77,89	4,01%	41,00%
20x10	6,18	67,37	3,25%	35,46%
20x20	5,86	55,80	3,08%	29,37%
50x5	17,18	536,66	1,40%	43,81%
50x10	20,06	460,28	1,64%	37,57%
50x20	19,38	414,57	1,58%	33,84%
100x5	35,21	2.266,32	0,71%	45,78%
100x10	42,59	1.969,09	0,86%	39,78%
100x20	45,30	1.760,82	0,92%	35,57%
200x10	92,39	8.192,97	0,46%	41,17%
200x20	95,12	7.326,13	0,48%	36,81%
500x20	257,15	46.272,65	0,21%	37,09%

6 Conclusões e Trabalhos Futuros

Neste trabalho foi desenvolvida uma heurística para o problema de *flow shop* com bloqueio para minimização do *makespan*. O método proposto faz uso de estruturas de vizinhança que incorporam propriedades de bloqueio já apresentadas na literatura e desenvolve novos métodos de aceleração para vizinhanças de troca e de inserção que consideram mais de uma tarefa. Além disso, foram introduzidos limitantes inferiores para avaliação de movimentos de vizinhanças do tipo *swap*.

Acredita-se que o método proposto é o primeiro baseado em uma meta-heurística *iterated local search* para o problema em questão. O algoritmo obteve melhores soluções do que as já encontradas na literatura em um total de 26 instâncias e empatou em 37 outras. Além disso, encontrou soluções melhores do que o DIWO proposto por Shao et al. [2019] em 35 instâncias distintas e empatou em 34 outras. Nas outras instâncias, obteve um *gap* mínimo inferior a 1% nos grupos de instâncias com menos de 500 tarefas. Os experimentos computacionais ainda demonstram que os limitantes inferiores propostos são capazes de evitar a avaliação de um número de soluções superior às propriedades clássicas já encontradas na literatura.

Como trabalhos futuros, sugere-se a incorporação de novos elementos ao problema, como tempos de *setup*, podendo-se adaptar os métodos de aceleração propostos nestas variantes. Ainda, testar o algoritmo proposto com maior tempo de execução deve acarretar na melhora dos *gaps* no grupo de instâncias de 500 tarefas.

Agradecimentos

Os autores agradecem a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concedido.

Referências

Bautista, J., Cano, A., Companys, R., e Ribas, I. (2012). Solving the fm—block—cmax problem using bounded dynamic programming. *Engineering Applications of Artificial Intelligence*, 25(6): 1235–1245.

Caraffa, V., Ianes, S., Bagchi, T. P., e Sriskandarajah, C. (2001). Minimizing makespan in a blocking flowshop using genetic algorithms. *International Journal of Production Economics*, 70(2):101–115.

Chen, H., Zhou, S., Li, X., e Xu, R. (2014). A hybrid differential evolution algorithm for a two-stage flow shop on batch processing machines with arbitrary release times and blocking. *International Journal of Production Research*, 52(19):5714–5734.

Davendra, D., Zelinka, I., Bialic-Davendra, M., Senkerik, R., e Jasek, R. (2012). Clustered enhanced differential evolution for the blocking flow shop scheduling problem. *Central European Journal of Operations Research*, 20(4):679–717.

Ding, J.-Y., Song, S., Gupta, J. N., Wang, C., Zhang, R., e Wu, C. (2016). New block properties for flowshop scheduling with blocking and their application in an iterated greedy algorithm. *International Journal of Production Research*, 54(16):4759–4772.

Grabowski, J. e Pempera, J. (2007). The permutation flow shop problem with blocking. a tabu search approach. *Omega*, 35(3):302–311.

Hall, N. G. e Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525.

Levner, E. (1969). Optimal planning of parts' machining on a number of machines. *Automatin and Remote Control*, 12(12):1972–1978.

Liu, S. Q. e Kozan, E. (2009). Scheduling a flow shop with combined buffer conditions. *International Journal of Production Economics*, 117(2):371–380.

McCormick, S. T., Pinedo, M. L., Shenker, S., e Wolf, B. (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37(6):925–935.

Miyata, H. H. e Nagano, M. S. (2019). The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Systems with Applications*, 137:130–156.

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

Ozolins, A. (2019). Improved bounded dynamic programming algorithm for solving the blocking flow shop problem. *Central European Journal of Operations Research*, 27(1):15–38.

Pan, Q.-K. e Wang, L. (2012). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega*, 40(2):218–229.

https://proceedings.science/p/157562?lang=pt-br

Pan, Q.-k., Wang, L., Sang, H.-y., Li, J.-q., e Liu, M. (2012). A high performing memetic algorithm for the flowshop scheduling problem with blocking. *IEEE Transactions on Automation Science and Engineering*, 10(3):741–756.

Pitty, S. S. e Karimi, I. A. (2008). Novel milp models for scheduling permutation flowshops. *Chemical Product and Process Modeling*, 3(1).

Prasad, S. D., Rajendran, C., e Chetty, O. K. (2006). A genetic algorithmic approach to multi-objective scheduling in a kanban-controlled flowshop with intermediate buffer and transport constraints. *The International Journal of Advanced Manufacturing Technology*, 29(5):564–576.

Ramalhinho Lourenço, H., Martin, O., e Stützle, T. (2019). Iterated Local Search: Framework and Applications. In Gendreau, M. e Potvin, J. Y., editors, *Handbook of Metaheuristics*, p. 129–168. Springer International Publishing.

Ribas, I., Companys, R., e Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, 39(3):293–301.

Ribas, I., Companys, R., e Tort-Martorell, X. (2015). An efficient discrete artificial bee colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications*, 42(15-16):6155–6167.

Ronconi, D. P. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, 87(1):39–48.

Ronconi, D. P. (2005). A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. *Annals of Operations Research*, 138(1):53–65.

Shao, Z., Pi, D., Shao, W., e Yuan, P. (2019). An efficient discrete invasive weed optimization for blocking flow-shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 78: 124–141.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285.

Toumi, S., Jarboui, B., Eddaly, M., e Rebaï, A. (2017). Branch-and-bound algorithm for solving blocking flowshop scheduling problems with makespan criterion. *International Journal of Mathematics in Operational Research*, 10(1):34–48.

Wang, L., Pan, Q.-K., Suganthan, P. N., Wang, W.-H., e Wang, Y.-M. (2010). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research*, 37(3):509–520.

Wang, X. e Tang, L. (2012). A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. *Applied Soft Computing*, 12(2):652–662.

Zhang, C., Xie, Z., Shao, X., e Tian, G. (2015). An effective vnssa algorithm for the blocking flowshop scheduling problem with makespan minimization. In *2015 International Conference on Advanced Mechatronic Systems (ICAMechS)*, p. 86–89. IEEE.

https://proceedings.science/p/157562?lang=pt-br