LVI SBPO SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
DE 04 ATÉ 07 NOVEMBRO 2024
FORTALEZA | CEARÁ

Heurísticas construtivas e busca local para o problema job shop flexível com flexibilidade de sequência e efeito de aprendizado

Kennedy A. G. Araújo
Departamento de Matemática Aplicada, IME-USP, Universidade de São Paulo
Rua do Matão, 1010, Cidade Universitária, 05508-090 - São Paulo - Brasil
kennedy94@ime.usp.br

Ernesto G. Birgin
Departamento de Ciência da Computação, IME-USP, Universidade de São Paulo
Rua do Matão, 1010, Cidade Universitária, 05508-090 - São Paulo - Brasil
egbirgin@ime.usp.br

Débora P. Ronconi
Departamento de Engenharia de Produção, EPUSP, Universidade de São Paulo
Av. Prof. Almeida Prado, 128, Cidade Universitária, 05508-900, São Paulo SP, Brasil
dronconi@usp.br

RESUMO

Este artigo explora o problema job shop flexível com flexibilidade de sequência e efeito de aprendizado (FJSSFLE). Apresentamos duas heurísticas construtivas baseadas em regras de despacho juntamente com uma busca local que usa uma vizinhança reduzida para resolver o problema. Apresentamos experimentos computacionais que mostram sua eficácia para achar boas soluções para instâncias de pequeno e grande porte do FJSSFLE, e também evidencia vantagens e desvantagens de cada método.

PALAVRAS CHAVE. Job shop flexível, Heurísticas Construtivas, Regras de despacho, Busca Local.

Otimização Combinatória

ABSTRACT

This article explores the flexible job shop problem with sequence flexibility and learning effect (FJSSFLE). We present two constructive heuristics based on dispatching rules along with a local search that uses a reduced neighborhood to solve the problem. We present computational experiments that demonstrate their effectiveness in finding good solutions for small and large instances of the FJSSFLE, and also highlight the advantages and disadvantages of each method.

KEYWORDS. Job shop scheduling problem, Constructive Heuristics, Dispaching rules, Local Search.

Combinatorial Optimization

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

1. Introdução

O job shop flexível (FJS) com flexibilidade de sequenciamento é um ambiente de produção com uma ampla gama de aplicações práticas relevantes, especialmente na indústria de impressão sob demanda atualmente [Lunardi et al., 2020, 2021]. Hoje, empresas no ramo de impressão sob demanda devem lidar com produção personalizada e priorizar a entrega pontual em um esforço para atender às necessidades de seus clientes. Nesse contexto, as atividades de produção são organizadas em máquinas flexíveis para gerenciar melhor a execução da ampla gama de tarefas demandadas. Outros ramos que se encaixam nesse ambiente de produção incluem a indústria de vidro [Alvarez-Valdes et al., 2005], a indústria de moldes [Gan e Lee, 2002], o agendamento de operações de suporte de aeronaves em decks de voo [Yu et al., 2017], o agendamento de pedidos de reparo em oficinas de reparo de colisão automotiva [Andrade-Pineda et al., 2020] e a construção de programas de produção para a produção de aço [De Moerloose e Maenhout, 2023]. Portanto, é importante que os métodos de resolução estejam preparados para lidar com as mais diversas características dos problemas reais encontrados neste ambiente de produção. Um desses fatores é o efeito de aprendizado, ou seja, como o tempo de processamento de uma operação varia com o número de vezes que é executada. Naturalmente, o uso de tempos de processamento que não são totalmente consistentes com a realidade pode levar a cronogramas imprecisos e resultar em perdas econômicas significativas.

O problema FJS é uma extensão do problema clássico job shop (JS), no qual cada operação pode ser processada por uma entre um conjunto de máquinas, em vez de uma única máquina. Essa característica é conhecida como flexibilidade de roteamento. Duas características adicionais são consideradas no presente trabalho: flexibilidade de sequenciamento e efeito de aprendizado. No FJS sem flexibilidade de sequenciamento, existe o conceito de uma tarefa, que consiste em um conjunto de operações que devem respeitar uma ordem sequencial de execução (primeiro a primeira operação, depois a segunda, depois a terceira, etc). A flexibilidade de sequenciamento consiste em considerar que as precedências entre as operações de uma mesma tarefa são dadas por um grafo acíclico direcionado (DAG) arbitrário. Em um problema de agendamento clássico, dada uma operação e uma máquina que pode processar essa operação, é dado um tempo de processamento fixo que corresponde ao tempo demandado pela máquina para processar a operação. O efeito de aprendizado corresponde ao adicional da vida real que consiste no fato de que uma pessoa aprende através da execução de uma tarefa repetitiva e, quanto mais vezes a executam, mais rápido o fazem. Neste trabalho, consideramos uma função de aprendizado que depende da posição que uma operação ocupa dentro da lista de operações executadas por uma máquina, ou seja, uma função de efeito de aprendizado baseada na posição com o objetivo de minimizar o maior tempo de completação, i.e. o *makespan*.

O restante deste artigo está organizado da seguinte forma. Na Seção 2, construímos soluções viáveis para o problema, que podem ser representadas por DAG, através de heurísticas construtivas. Na Seção 3, introduzimos o conceito de vizinhança, analisamos diferentes estratégias para a busca local. A Seção 4 é dedicada a experimentos numéricos com os métodos propostos. Conclusões e direções futuras de trabalho são apresentadas na seção final.

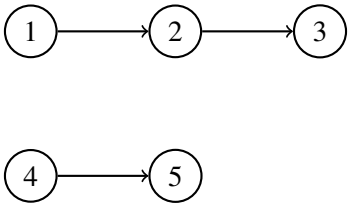
2. Heurísticas Construtivas

Os dados de uma instância do FJSSFLE baseado na posição consistem em (a) um conjunto de operações $\mathcal{O}$ e um conjunto de máquinas $\mathcal{F}$; (b) para cada operação $i \in \mathcal{O}$, um subconjunto $\mathcal{F}i \subseteq \mathcal{F}$ contendo as máquinas que podem processar i; (c) para cada par operação-máquina (i, k) com $i \in \mathcal{O}$ e $k \in \mathcal{F}i$, um tempo de processamento padrão p_{ik}; e (d) um conjunto de arcos $\widehat{A} \subseteq \mathcal{O} \times \mathcal{O}$ representando as relações de precedência entre as operações. O efeito de aprendizado é dado por uma função $\psi_\alpha(p, r)$ que, dado um tempo de processamento padrão p e uma posição r, retorna

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

o tempo de processamento real de uma operação com tempo de processamento padrão p quando processada na r-ésima posição de uma máquina. O parâmetro $\alpha > 0$ representa a taxa de efeito de aprendizado. No trabalho atual, consideramos $\psi_\alpha(p, r) = \lfloor 100 \cdot p/r^\alpha + 1/2 \rfloor$. Um exemplo simples de uma instância é mostrado na Figura 1.

Uma solução viável para uma instância do FJS com flexibilidade de sequenciamento e efeito de aprendizado baseado na posição pode ser representada por um DAG $G = (V, A)$ conforme mostrado na Figura 2a. Esse grafo é às vezes referido como grafo de solução na literatura. Os vértices de G, representados pelo conjunto V, correspondem às operações mais os vértices fictícios s e t, ou seja, $V = \mathcal{O} \cup \{s, t\}$. Os arcos, representados pelo conjunto A, correspondem aos arcos em $\widehat{A}$ representando as relações de precedência entre operações (em preto na figura), arcos que saem de s para operações que não têm predecessores e arcos que vão para t de operações que não precedem nenhuma outra operação (em roxo na figura). Arcos que partem de s e arcos que chegam em t são chamados de arcos fictícios. Além disso, os arcos tracejados representam a atribuição de operações a máquinas e a ordem em que as operações são processadas por cada máquina. Esses arcos são chamados de arcos de máquina. Cada nó $i \in V \setminus \{s, t\} = \mathcal{O}$, ou seja, cada operação, tem um valor w_i associado a ele que representa seu tempo de processamento real, que é calculado com a função de aprendizado usando o tempo de processamento padrão da operação e a posição que a operação ocupa na máquina à qual foi atribuída. Os nós s e t estão associados ao valor zero. O caminho mais longo entre os nós s e t é chamado de caminho crítico (destacado em amarelo na figura), e seu comprimento corresponde ao tempo de conclusão (makespan) da solução representada.



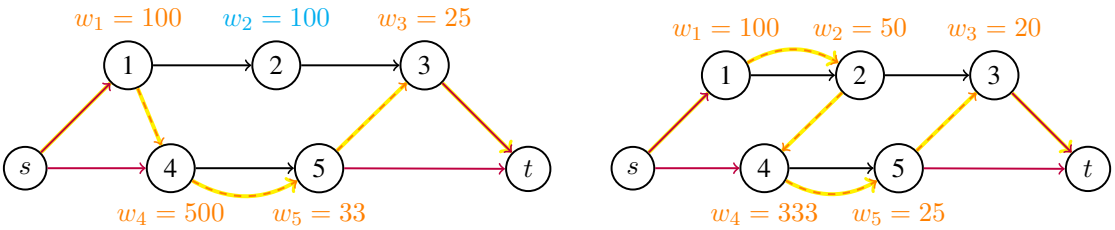
		Machines	
		1	2
Operations	1	1	1
	2	1	1
	3	1	1
	4	10	10
	5	1	1

Figura 1: À esquerda, representação das restrições de precedência das operações por um DAG $D = (\mathcal{O}, \widehat{A})$, onde $\mathcal{O} = 1, 2, \ldots, 5$ representa o conjunto de operações e $\widehat{A} = (1, 2), (2, 3), (4, 5)$ é o conjunto de arcos que representa as restrições de precedência. Neste exemplo simples, as restrições de precedência são dadas por uma ordem linear, ou seja, não há flexibilidade de sequenciamento. Esta instância tem duas máquinas e cada uma das cinco operações pode ser processada em qualquer uma das duas máquinas, ou seja, $\mathcal{F} = 1, 2$ e $\mathcal{F}_i = \mathcal{F}$ para todos $i \in \mathcal{O}$. Isso significa que há flexibilidade de roteamento completa. A tabela à direita mostra os tempos de processamento padrão p_{ik} das cinco operações em cada uma das duas máquinas.

Nesta seção, propomos duas heurísticas construtivas para o FJSSFLE. Heurísticas construtivas são algoritmos que constroem uma solução viável do zero, selecionando e sequenciando iterativamente uma operação de cada vez. As duas heurísticas construtivas propostas são baseadas na regra de tempo de início mais cedo (EST) [Birgin et al., 2014] e na regra de tempo de conclusão mais cedo (ECT) [Leung et al., 2005]. O objetivo é utilizá-las para fornecer uma solução viável inicial para uma busca local que será usada para resolver um conjunto de instâncias de teste. A solução final é representada por um grafo direcionado acíclico (DAG), adaptado de Mastrolilli e Gambardella [2000].

O Algoritmo 1 apresenta a heurística construtiva baseada na regra de tempo de início mais cedo (EST). Em que, a cada iteração, a operação que pode ser executada mais cedo será alocada. No

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br



(a) (b)

Figura 2: Nesta figura, consideramos a instância na Figura 1 com taxa de aprendizado $\alpha = 1$. O digrafo à esquerda (Figura 2a) representa uma solução viável na qual a máquina 1 (associada à cor ciano) processa apenas a operação 2, enquanto a máquina 2 (associada à cor laranja) processa as operações 1, 4, 5 e 3 nessa ordem. Os números coloridos representam o tempo de processamento real das operações, com a influência do efeito de aprendizado. O caminho crítico, cujo comprimento corresponde ao makespan, é dado pelo caminho $s, 1, 4, 5, 3, t$ (destacado em amarelo na imagem). O digrafo à direita (Figura 2b) representa a solução viável obtida realocando a operação 2, que não estava no caminho crítico, da máquina 1 para a máquina 2 entre as operações 1 e 4. A solução viável construída, com caminho crítico dado por $s, 1, 2, 4, 5, 3, t$, tem um makespan menor que o original (528 contra 658).

algoritmo, temos como entrada o conjunto de operações, $\mathcal{O}$, o conjunto de máquinas, $\mathcal{F}$, matriz dos tempos de processamento p e conjunto de arcos de precedência $\hat{A}$. O valor r_v^{op} se refere ao tempo que a operação v está pronta para ser processada, w_v se refere ao seu tempo de processamento atual e c_v ao seu tempo de conclusão. Do lado das máquinas, r_k^{mac} representa o instante em que a máquina k é liberada e g_k representa sua primeira posição livre, que é aquela que seria ocupada se uma operação fosse atribuída a ela (ambas as quantidades se referem ao sequenciamento parcial sendo construído). f_v indicará a qual máquina a operação v foi atribuída e cada máquina k terá uma lista ordenada Q_k com a sequência de operações a serem processadas. Após as inicializações (linhas 2 a 5), vem o loop principal, que é executado enquanto ainda houver operações não sequenciadas. Entre as não sequenciadas, o tempo em que elas estão disponíveis é calculado para todas aquelas que já têm todas as operações precedentes agendadas (linhas 7 a 9). Na linha 10, observando os tempos em que as operações e das máquinas estariam disponíveis, é calculado o instante mais cedo $r_{\min}$ em que uma operação poderia ser sequenciada e o conjunto E de pares operação/máquina que poderiam começar naquele instante $r_{\min}$ é construído. Como observado em [Birgin et al., 2014], $|E|$ pode ser bastante grande e uma regra de desempate pode melhorar significativamente o desempenho do método. Assim, na linha 11, entre todos os pares operação/máquina em E, levando em consideração o efeito de aprendizado, o par $(\hat{v}, \hat{k})$ com o tempo de processamento mais curto é par escolhido. Na linha 12, $w_{\hat{v}}$, $f_{\hat{v}}$ e $c_{\hat{v}}$ são definidos e o tempo $r_{\hat{k}}^{\text{mac}}$ e a posição livre $g_{\hat{k}}$ da máquina $\hat{k}$ são atualizados. Nas linhas 13 e 14, o arco da máquina correspondente é inserido no grafo G (o arco não deve ser inserido se a operação $\hat{v}$ for a primeira da máquina $\hat{k}$). Por fim, a lista de operações atribuídas à máquina $\hat{k}$ é atualizada e a operação agendada é removida do conjunto de operações ainda não agendadas. Depois que todas as operações foram agendadas, o caminho crítico em G é calculado (linha 16) para determinar o valor de makespan $C_{\max}$. Isso é feito com uma adaptação do algoritmo de Bellman-Ford para caminho máximo em $O(|\mathcal{O}|)$. As inicializações nas linhas de 2 a 5 do Algoritmo 1 têm complexidade $O(|\mathcal{O}| + |\widehat{\mathcal{A}}| + |\mathcal{F}|)$. Dentro do loop principal (linhas de 6 a 15), as linhas de 7 a 9 têm complexidade $O(|\mathcal{A}|) = O(|\widehat{A}| + |\mathcal{O}| + |\mathcal{F}|)$, as linhas de 10 a 11 têm complexidade $O(|\mathcal{O}| + \sum_{i \in \mathcal{O}} |F_i|)$, e a linha 12 tem complexidade $O(|\mathcal{O}|)$. Como o loop principal é executado $|\mathcal{O}|$ vezes, sua complexidade total é $O(|\mathcal{O}|(|\widehat{A}| + |\mathcal{F}| + \sum_{i \in \mathcal{O}} |\mathcal{F}_i|))$.

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

Como a complexidade do loop principal é maior que a complexidade da inicialização e da linha 16, então a complexidade do Algoritmo 1 é dada pela complexidade de seu loop principal. É importante notar que $\gamma = \sum_{i \in \mathcal{O}} |\mathcal{F}_i|$ está entre $|\mathcal{O}|$ e $|\mathcal{O}||\mathcal{F}|$, mas preferimos manter a complexidade expressa como uma função de γ porque γ é uma medida do tamanho da entrada que depende da flexibilidade de sequenciamento da instância em consideração. Também é importante notar que a complexidade do algoritmo depende da flexibilidade de roteamento das operações, ou seja, depende do número de relações de dependência em $\widehat{A}$. Portanto, é importante representar uma instância de forma que $\widehat{A}$ corresponda a uma redução transitiva do digrafo de precedências.

Algorithm 1: Computa um grafo de solução $G = (V, A)$, f, Q, e w usando a heurística EST. Então, em G, computa o maior caminho $\mathcal{P}$ de s a t e seu comprimento $C_{\max}$.

Input: $\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$
Output: f, w, Q, $G = (V, A)$, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ
1 **function** EST($\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$, f, w, Q, G, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ)
2 Atribua $A \leftarrow \widehat{A} \cup \{(s, j) \mid (\cdot, j) \notin \widehat{A}\} \cup \{(i, t) \mid (i, \cdot) \notin \widehat{A}\}$ e defina $V := \mathcal{O} \cup \{s, t\}$ e $G = (V, A)$.
3 Atribua $r_v^{\text{op}} \leftarrow +\infty$ para todo $v \in V$ e defina $r_s^{\text{op}} := 0$, $w_s := w_t := 0$, e $c_s := 0$.
4 Atribua $r_k^{\text{mac}} \leftarrow 0$ e $g_k \leftarrow 1$ para todo $k \in \mathcal{F}$.
5 Inicie $\Pi \leftarrow V \setminus \{s, t\}$ como o conjunto de operações não sequenciadas, e Q_k como uma lista vazia para todo $k \in \mathcal{F}$.
6 **while** $\Pi \neq \emptyset$ **do**
7 **for** $v \in \Pi$ **do**
8 **if** $\Pi \cap \{i \mid (i, v) \in A\} = \emptyset$ **then**
9 $r_v^{\text{op}} \leftarrow \max\{c_i \mid i \in V \setminus \Pi \text{ tal que } (i, v) \in A\}$
10 Atribua $r_{\min} = \min\{\max(r_v^{\text{op}}, r_k^{\text{mac}}) \mid v \in \Pi, k \in \mathcal{F}_v\}$ e seja E o conjunto dos pares (v, k) com $v \in \Pi$ e $k \in \mathcal{F}_v$ tal que $\max(r_v^{\text{op}}, r_k^{\text{mac}}) = r_{\min}$.
11 $(\hat{v}, \hat{k}) \leftarrow \text{argmin}\{r_{\min} + \psi_\alpha(p_{v,k}, g_k) \mid (v, k) \in E\}$.
12 Defina $w_{\hat{v}} := \psi_\alpha(p_{\hat{v},\hat{k}}, g_{\hat{k}})$, $f_{\hat{v}} := \hat{k}$ e $c_{\hat{v}} := \max(r_{\hat{v}}^{\text{op}}, r_{\hat{k}}^{\text{mac}}) + w_{\hat{v}}$, e atribua $r_{\hat{k}}^{\text{mac}} \leftarrow c_{\hat{v}}$ e $g_{\hat{k}} \leftarrow g_{\hat{k}} + 1$.
13 **if** $|Q_{\hat{k}}| \neq 0$ **then**
14 Seja $Q_{\hat{k}} = i_1, \ldots, i_{|Q_{\hat{k}}|}$. Atribua $A \leftarrow A \cup \{(i_{|Q_{\hat{k}}|}, \hat{v})\}$.
15 Insira $\hat{v}$ no final de $Q_{\hat{k}}$ e atribua $\Pi \leftarrow \Pi \setminus \{\hat{v}\}$.
16 CaminhoCritico($\mathcal{F}$, f, w, Q, G, $\mathcal{U}$, $\mathcal{P}$, $C_{\max}$, τ).

O caminho crítico no grafo direcionado $G = (V, A)$ pode ser calculado com uma adaptação [Cormen et al., 2022, §22.2] do algoritmo de Bellman-Ford em $O(|V|+|A|)$. Além do caminho crítico $\mathcal{P}$, o algoritmo deve retornar uma ordenação topológica $\mathcal{U}$ dos vértices de G e um vetor τ de dimensão $|\mathcal{F}|$. O vetor τ armazena, no elemento τ_k, a maior posição na lista Q_k (lista de operações atribuídas à máquina k) que contém uma operação no caminho crítico.

O algoritmo para a heurística construtiva baseada na regra ECT é muito semelhante ao Algoritmo 1, exceto por um detalhe. Na heurística construtiva baseada no EST, primeiro calculamos o instante $r_{\min}$, que é o instante mais cedo em que uma operação não agendada poderia ser

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

iniciada. Todos os pares operação/máquina que poderiam começar nesse instante são considerados, e o par com o menor tempo de processamento é selecionado. Mas, como todos começariam no instante $r_{\min}$, dizer que o par com o menor tempo de processamento é escolhido é o mesmo que dizer que o par que termina mais cedo é selecionado. Essa é a ideia que é levada ao extremo na heurística construtiva baseada na regra ECT: sem limitar a escolha aos pares operação/máquina que poderiam começar o mais cedo possível, escolhemos o par operação/máquina que terminará mais cedo, mesmo que o processamento da operação não comece o mais cedo possível. A complexidade de tempo no pior caso do algoritmo para a heurística ECT é a mesma que a do Algoritmo 1.

A heurística baseada em EST dá prioridade aos pares operação/máquina que podem começar mais cedo. No início da construção, isso corresponde, aproximadamente, a dar prioridade a todas as primeiras operações de cada tarefa, que são operações que não têm precedentes (operações 1 e 4 no exemplo da Figura 1). Ainda assim, devido à intenção de agendar operações o mais cedo possível, é possível que a preferência seja dada a máquinas vazias, construindo soluções que usam várias máquinas. Ao agendar rapidamente as primeiras operações de cada trabalho, mais operações têm seus precedentes agendados, aumentando o número de possibilidades (espaço de busca) nas iterações futuras do método. Por outro lado, a heurística baseada na regra ECT escolhe os pares operação/máquina que terminam mais cedo, independentemente de serem aqueles que podem começar mais cedo ou não. Essa estratégia pode limitar o número de pares operação/máquina disponíveis em iterações futuras, reduzindo o espaço de busca do método. Além disso, a escolha pelo par operação/máquina que pode terminar mais cedo, combinada com o efeito de aprendizado, leva o método a agendar operações em máquinas que já têm várias operações atribuídas a elas, já que quanto maior a posição na máquina, menor o tempo de processamento (reduzido pelo efeito de aprendizado baseado em posição). Isso leva à construção de soluções em que nem todas as máquinas são usadas. Dependendo da taxa de aprendizado α considerada e da densidade do DAG de precedências da instância em questão, uma heurística pode ser melhor que a outra.

3. Vizinhança Reduzida e Busca Local

Dada uma solução viável e um DAG $G = (V, A)$ que a representa, uma nova solução viável pode ser construída removendo uma operação da máquina à qual foi atribuída e reinserindo-a na mesma máquina, mas em outra posição ou em outra máquina. Quando uma operação é removida, os arcos de máquina adjacentes a ela devem ser removidos e um novo arco indo da operação anterior para a seguinte à removida (se ambas existirem) deve ser criado. Quando a operação é reinserida, uma operação reversa similar também deve ser feita. Ao reinserir a operação, é importante verificar se um ciclo não é produzido no digrafo. Somente reinserções que não criam ciclos constroem um digrafo que corresponde a uma solução viável. Quando não há efeito de aprendizado, é conhecido [Mastrolilli e Gambardella, 2000] que há chances de construir uma solução viável melhor removendo e realocando operações que fazem parte do caminho crítico apenas. Se uma operação não faz parte do caminho crítico, sua remoção e reinserção não pode diminuir o comprimento do caminho crítico. Pode aumentá-lo ou criar outro caminho ainda mais longo. Isso é falso ao considerar o efeito de aprendizado. Um exemplo é mostrado na Figura 2b.

Dada uma solução viável, podemos definir sua vizinhança como o conjunto de todas as soluções viáveis que podem ser obtidas removendo e reinserindo uma única operação. Quando não há efeito de aprendizado, apenas a remoção e reinserção de operações do caminho crítico podem levar a vizinhos melhores, ou seja, vizinhos com menor makespan. Este fato é amplamente utilizado para gerar apenas vizinhos promissores. A observação no parágrafo anterior mostra que essa redução não pode ser usada no problema que estamos considerando no presente trabalho. Isso nos leva a analisar se toda remoção e reinserção que não gera ciclos tem o potencial de gerar um vizinho com makespan menor ou se qualquer redução de vizinhança é possível. O ponto principal é obser-

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

var que quando uma operação é removida de uma máquina, as operações que estavam sequenciadas para serem processadas posteriormente nessa máquina têm sua posição diminuída em uma unidade e, consequentemente, seu tempo de processamento real aumentado. Da mesma forma, na máquina onde a operação é inserida, as operações programadas para serem processadas após a operação inserida têm sua posição aumentada em uma unidade e, portanto, seu tempo de processamento é diminuído. Essas modificações podem mudar o makespan, seja melhorando ou piorando.

Considere uma solução viável representada por um DAG $G = (V, A)$. Seja $\mathcal{P}$ um caminho crítico em G, com comprimento $C_{\max}$. Seja $i \in \mathcal{O}$ uma operação arbitrária. Chamamos de f_i a máquina à qual i está atribuída. Seja $k \in \mathcal{F}$ uma máquina arbitrária. Chamamos $Q_k = i_1, \ldots, i_{|Q_k|}$ a lista ordenada de operações atribuídas à máquina k. Se uma operação i está atribuída à máquina f_i e está na posição γ de Q_{f_i}, então seu tempo de processamento real é dado por $w_i = \psi_\alpha(p_{i,f_i}, \gamma)$. Pretendemos calcular todos os vizinhos da solução representada por G, f, Q e w. Os vizinhos serão construídos, para todo $v \in \mathcal{O}$, removendo v e reinserindo v em todos os lugares possíveis que não gerem um ciclo. Queremos determinar se existem inserções que podem ser ignoradas porque sabemos *a priori* que não levarão a uma redução no makespan.

Seja $v \in \mathcal{O}$ uma operação arbitrária. O cálculo dos vizinhos da solução atual (associados à remoção e reinserção de v) começa calculando um digrafo $G_v^- = (V^-, A^-)$ no qual a operação v é removida. Esse grafo é às vezes referido como um grafo reduzido na literatura. As quantidades f^-, Q^- e w^- associadas a G_v^- também são calculadas. Esse digrafo com suas informações associadas é uma estrutura intermediária necessária para o cálculo dos vizinhos e, como a operação v não está atribuída a nenhuma máquina, não representa uma solução viável. Essa tarefa é implementada no Algoritmo *RemoveOp*. Além disso o conjunto $\mathcal{R}_v^{\leftarrow}$ de vértices que alcançam v e o conjunto de vértices $\mathcal{R}_v^{\rightarrow}$ que são alcançados a partir de v em G_v^- são calculados, o que será útil para detectar ciclos em futuras reinserções de v. O caminho mais longo $\mathcal{P}^-$ no digrafo G_v^- é calculado, o que será útil para determinar se uma reinserção tem chances de reduzir o makespan ou não. Chamamos ξ de comprimento de $\mathcal{P}^-$. (Não o chamamos de $C_{\max}^-$ porque como G_v^- não representa uma solução viável, então o comprimento do caminho $\mathcal{P}^-$ não representa um makespan.) Junto com o cálculo de $\mathcal{P}^-$, o algoritmo também calcula, para cada máquina k, a menor posição τ_k tal que, para todo $\gamma > \tau_k$, a γ-ésima operação processada pela máquina k não está em $\mathcal{P}^-$. (Se a máquina k não processa nenhuma operação em $\mathcal{P}^-$, então $\tau_k = 0$.)

Seja G o digrafo, com as quantidades associadas f, Q e w, representando a solução viável atual. Seja $\mathcal{P}$ o caminho crítico em G, com comprimento $C_{\max}$. Seja v a operação que removemos e desejamos reinserir. Seja G_v^- o digrafo com v removido e deixe f^-, Q^- e w^- serem as quantidades associadas a G_v^-. Seja $\mathcal{P}^-$ o caminho crítico em G_v^-, com comprimento ξ, e, para cada máquina k, deixe τ_k ser a menor posição em Q_k tal que cada operação em uma posição após τ_k não esteja em $\mathcal{P}^-$. Deixe κ ser uma máquina e γ ser uma posição na lista Q_κ^- tal que inserir v na posição γ de Q_κ^- não gere um ciclo. Tal inserção tem chance de gerar um novo digrafo cuja solução viável associada tenha um makespan menor que $C_{\max}$? Se $\xi \geq C_{\max}$ e $\gamma > \tau_k$, então a resposta é *não*. Isso ocorre porque o caminho $\mathcal{P}^-$ com comprimento ξ não menor que $C_{\max}$ já existe e a inserção de v na máquina κ, em uma posição γ posterior a τ_k, não modificará o tempo de processamento real de nenhuma operação em $\mathcal{P}^-$. Se $\xi < C_{\max}$ ou $\xi \geq C_{\max}$, mas $\gamma \leq \tau_k$, então as chances existem.

Deve-se notar que, estritamente falando, o fato de v estar em $\mathcal{P}$ ou não não está relacionado à resposta à pergunta acima. Mas antecipando algo que virá mais tarde, como a redução de vizinhança orientada pela resposta à pergunta pode ser bastante pequena, consideraremos nos experimentos, de forma heurística, $v \in \mathcal{P}$ como equivalente ou fortemente correlacionado a $\xi < C_{\max}$. Ou seja, consideraremos que remover uma operação do caminho crítico provavelmente implicará em $\xi < C_{\max}$. Isso é muito plausível para valores moderados do fator de aprendizado α, nos quais

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

uma possível redução de uma unidade na posição da máquina de algumas operações do caminho crítico não anula o benefício de remover uma operação do caminho crítico.

A tarefa de reinserir v em G_v^- na posição γ da máquina κ gera um DAG que chamamos de G_v^+. Essa tarefa é semelhante à tarefa de remoção. A construção de G_v^+, suas quantidades associadas f^+, Q^+ e w^+, e seu caminho crítico $\mathcal{P}^+$ com comprimento $C_{\max}^+$ é implementada no InsereOp. O Algoritmo 2 implementa uma busca local pelo melhor vizinho com a redução de vizinhança já discutida. Ele corresponde a uma busca local clássica com uma estratégia de melhor vizinho (*best improvement*). O único detalhe relevante que resta a ser explicado é como determinar se uma inserção gera um ciclo ou não. Um ciclo será criado em G_v^+ apenas se v for inserido em uma posição que deixe algum $u \in \mathcal{R}_v^{\leftarrow}$ para ser processado após v na máquina κ ou algum $u \in \mathcal{R}_v^{\rightarrow}$ para ser processado antes de v na máquina κ. Os limites $\underline{\gamma}$ e $\bar{\gamma}$ tais que $\underline{\gamma}+1 \leq \gamma \leq \bar{\gamma}$ evitam ciclos, são calculados nas linhas 7 e 8. Uma possível redução desse intervalo é calculada nas linhas 9 e 10, eliminando a possibilidade de fazer inserções após τ_κ se $\xi \geq C_{\max}$, como já discutido.

Algorithm 2: Busca local com vizinhança reduzida e estratégia de best improvement.

Input: $\mathcal{O}, \mathcal{F}, p, G=(V,A), f, w, Q, \mathcal{P}, C_{\max}$
Output: $G^\star=(V^\star,A^\star), f^\star, w^\star, Q^\star, \mathcal{P}^\star, C_{\max}^\star$

1 **function** BuscaLocal($\mathcal{O}, \mathcal{F}, p, G, f, w, Q, \mathcal{P}, C_{\max}, G^\star, f^\star, w^\star, Q^\star, \mathcal{P}^\star, C_{\max}^\star$)
2 **do**
3 $C_{\max}^{\text{bn}} \leftarrow +\infty$
4 **for** $v \in \mathcal{O}$ **do**
5 RemoveOp($\mathcal{O}, p, v, f, Q, w, G, f^-, Q^-, w^-, G_v^-, \mathcal{P}^-, \xi, \mathcal{R}_v^{\leftarrow}, \mathcal{R}_v^{\rightarrow}, \tau$)
6 **for** $k \in \mathcal{F}_v$ **do**
7 Assuma $\underline{\gamma}$ a posição da última operação em $Q_k^- = i_1, \ldots, i_{|Q_k^-|}$ tal que $i_{\underline{\gamma}} \in \mathcal{R}_v^{\leftarrow}$ e assuma $\underline{\gamma}=0$ se $i_\ell \notin \mathcal{R}_v^{\leftarrow}$ para todo $\ell = 1, \ldots, |Q_k^-|$.
8 Assuma $\bar{\gamma}$ a posição da primeira operação em $Q_k^- = i_1, \ldots, i_{|Q_k^-|}$ tal que $i_{\bar{\gamma}} \in \mathcal{R}_v^{\rightarrow}$ e assuma $\bar{\gamma}=|Q_k^-|+1$ se $i_\ell \notin \mathcal{R}_v^{\rightarrow}$ para todo $\ell = 1, \ldots, |Q_k^-|$.
9 **if** $\xi \geq C_{\max}$ **then**
10 $\bar{\gamma} \leftarrow \min\{\bar{\gamma}, \tau_k\}$, onde τ_k é tal qual não existe operação crítica após τ_k em Q_k^- ($\tau_k=0$ se não há operação crítica em Q_k^-).
11 **for** $\gamma = \underline{\gamma}+1, \ldots, \bar{\gamma}$ **do**
12 InsereOp($\mathcal{O}, p, v, \gamma, k, f^-, Q^-, w^-, G_v^-, f^+, Q^+, w^+, G_v^+, \mathcal{P}^+, C_{\max}^+$)
13 **if** $C_{\max}^+ < C_{\max}^{\text{bn}}$ **then**
14 $G^{\text{bn}}, f^{\text{bn}}, w^{\text{bn}}, Q^{\text{bn}}, \mathcal{P}^{\text{bn}}, C_{\max}^{\text{bn}} \leftarrow G_v^+, f^+, w^+, Q^+, \mathcal{P}^+, C_{\max}^+$
15 $\delta \leftarrow C_{\max} - C_{\max}^{\text{bn}}$
16 **if** $\delta > 0$ **then**
17 $G, f, w, Q, \mathcal{P}, C_{\max} \leftarrow G^{\text{bn}}, f^{\text{bn}}, w^{\text{bn}}, Q^{\text{bn}}, \mathcal{P}^{\text{bn}}, C_{\max}^{\text{bn}}$
18 **while** $\delta > 0$
19 $G^\star, f^\star, w^\star, Q^\star, \mathcal{P}^\star, C_{\max}^\star \leftarrow G, f, w, Q, \mathcal{P}, C_{\max}$

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

4. Experimentos Computacionais

Nesta seção, apresentamos experimentos numéricos. Primeiro, desejamos avaliar as duas heurísticas construtivas apresentadas. Segundo, desejamos avaliar diferentes estratégias para a busca local proposta e tentar inferir qual é mais eficaz na busca por soluções de melhor qualidade. Em todos os casos, consideramos as 50 instâncias introduzidas por Birgin et al. [2014] com a taxa de aprendizado $\alpha \in \{0,1; 0,2; 0,3\}$ para um total de 150 instâncias.

Os experimentos foram realizados em uma máquina com processador Intel i9-12900K (12ª geração) operando a 5,200GHz e 128 GB de RAM. As heurísticas construtivas e busca local foram implementadas na linguagem de programação C++. O código foi compilado usando g++ 10.2.1.

Na Tabela 1 o resumo dos resultados da avaliação duas heurísticas construtivas é apresentado. Para cada grupo instância e taxa de aprendizado, a média do makespan e número de vitórias, entre as soluções encontradas pelas duas heurísticas construtivas, são apresentados. Em todas as instâncias, as heurísticas construtivas levam menos de 0.001 segundos de tempo de CPU para construir uma solução. Para as instâncias testadas há uma clara vantagem da heurística construtiva EST nas instâncias do tipo DA, enquanto, por outro lado, há uma clara vantagem da heurística construtiva ECT nas instâncias do tipo Y. A estratégia gulosa de ECT de escolher o par operação/máquina que termina primeiro parece compensar em situações onde, porque já há pouca flexibilidade de sequenciamento, a escolha gulosa não causa uma grande diminuição do espaço de busca.

		DAFJS		YFJS	
		EST	ECT	EST	ECT
$\alpha = 0,1$	makespan	65.249,50	67.439,80	87.865,80	80.338,80
	#vitórias	20	10	5	15
$\alpha = 0,2$	makespan	54.310,83	57.746,00	74.373,85	68.682,05
	#vitórias	24	6	7	13
$\alpha = 0,3$	makespan	45.578,87	48.181,97	65.850,25	59.452,35
	#vitórias	28	2	7	13

Tabela 1: Tabela resumo dos experimentos computacionais para os valores de makespan para as instâncias de teste resolvidas com as heurísticas construtivas.

Avaliamos agora variações da busca local descrita no Algoritmo 2. No algoritmo, a busca local usa a estratégia de *best improvement* e faz uso da redução do vizinhança. Portanto, chamamos essa versão de "busca local com a estratégia de best improvement e vizinhança reduzida". A redução de vizinhança é implementada nas linhas 9 e 10. Se removermos essas duas linhas, obtemos uma versão que chamamos de "busca local com best improvement e vizinhança completa". A versão com vizinhança reduzida não considera vizinhos que são garantidamente piores do que a solução atual. Portanto, a solução obtida com vizinhança reduzida deve ser idêntica à solução obtida com o vizinhança completa. (Na verdade, todos as iterações das duas versões devem ser idênticos e não apenas a solução final). Apenas uma redução do tempo de CPU é esperada. Decidimos considerar ainda outra versão que apresentaria uma redução mais drástica no tempo de CPU, embora com possível perda de qualidade na solução. Chamamos essa versão de "busca local com best improvement e vizinhança cortada". Esta versão consiste em alterar $v \in \mathcal{O}$ para $v \in \mathcal{P}$ na linha 4 do Algoritmo 2. Ou seja, apenas as operações no caminho crítico são realocadas, uma vez que há uma maior tendência para essas realocações gerarem vizinhos de melhor qualidade. Temos então três versões diferentes da busca local com a estratégia de best improvement que são distinguíveis pela vizinhança usada: completa, reduzida e cortada. Cada uma delas corresponde a variações

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

mínimas do Algoritmo 2 conforme já descrito. Além disso, consideramos as mesmas três versões, mas usando a estratégia de interromper a busca da vizinhança ao encontrar o primeiro vizinho que melhora a solução atual, ou seja, a estratégia de *first improvement*. Essa mudança corresponde, no Algoritmo 2, a interromper o laço da linha 4 na primeira vez que a linha 16 é executada.

Os resultados das seis variações da busca local aplicadas às 150 instâncias consideradas são mostrados na Tabela 2. Nas tabelas, mostramos o makespan médio das soluções obtidas, o número de iterações médio que a busca local fez até encontrar uma solução que é melhor do que todos os seus vizinhos (este é o critério de parada conforme descrito no Algoritmo 2), e o tempo médio de CPU em segundos. A tabela também não mostra nada relacionado à vizinhança completa. O que deve ser dito sobre o uso da vizinhança completa é que, em todas as instâncias, como esperado, a solução obtida foi idêntica à solução obtida com a vizinhança reduzida, o número de iterações também foi o mesmo, e a vizinhança reduzida promoveu uma redução de 52,51% no tempo de CPU.

Quando comparamos as estratégias de first e best improvement, os resultados são bastante semelhantes, mas a estratégia do melhor aprimoramento sempre encontra soluções de melhor qualidade usando menos tempo de CPU. Especificamente, a estratégia de best improvement retorna soluções que são, em média, 1,02% e 0,70% melhores do que as soluções retornadas pela estratégia de primeira melhoria, quando consideramos as vizinhanças reduzida e cortada, respectivamente. Portanto, daqui em diante, focamos em avaliar a vizinhança reduzida e a vizinhança cortada à estratégia do melhor aprimoramento.

A vizinhança cortada elimina, em média, 90,34% das soluções da vizinhança reduzida, promovendo uma redução proporcional no tempo de CPU. No entanto, adotar a vizinhança cortada pode levar a uma perda de qualidade na solução final obtida pelo método de busca local. Em média, quando comparado com a busca local com vizinhança reduzida, a busca local com a vizinhança cortada encontra soluções com um makespan 0,69% pior. Quando comparamos a solução final com a solução inicial, a busca local usando a vizinhança reduzida melhora a solução inicial em, em média, 6,88%, enquanto a busca local usando a vizinhança cortada melhora a solução inicial em 6,11%. Em conclusão, a busca local com a vizinhança cortada é significativamente mais rápida do que a busca local com a vizinhança reduzida e encontra soluções apenas ligeiramente piores do que as soluções encontradas por esta última.

		DAFJS				YFJS			
		First Improvement		Best Improvement		First Improvement		Best Improvement	
		Cortada	Reduzida	Cortada	Reduzida	Cortada	Reduzida	Cortada	Reduzida
$\alpha = 0,1$	makespan	59.008,00	58.925,53	58.305,43	57.974,70	71.528,60	71.895,70	71.227,20	70.704,30
	#iterações	27,73	61,37	14,53	20,90	18,15	57,1	8,30	19,60
	tempo	0,009	0,070	0,016	0,073	0,011	0,264	0,012	0,243
	#vitórias	3	2	7	10	5	7	4	11
$\alpha = 0,2$	makespan	50.713,70	50.400,03	49.944,33	49.835,73	62.812,75	62.184,05	62.154,65	61.446,60
	#iterações	18,83	39,4	12,03	15,90	14,05	51	7,70	17,20
	tempo	0,005	0,051	0,011	0,051	0,009	0,309	0,010	0,225
	#vitórias	3	8	7	13	5	8	7	12
$\alpha = 0,3$	makespan	42.854,47	42.315,00	42.383,17	42.018,73	55.087,10	54.328,30	54.883,00	53.749,15
	#iterações	17,3	36,23	9,33	14,77	10,05	38,9	6,35	16
	tempo	0,006	0,048	0,009	0,054	0,007	0,208	0,009	0,201
	#vitórias	2	8	9	15	6	10	7	14

Tabela 2: Resumo dos experimentos com as estratégias e vizinhanças usadas para a busca local usando as instâncias propostas por Birgin et al. [2014] com taxa de aprendizagem $\alpha \in \{0,1; 0,2; 0,3\}$.

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

5. Conclusões

Neste trabalho, abordamos o problema de sequenciamento do job shop flexível com flexibilidade de sequenciamento e efeito de aprendizado baseado na posição. Utilizamos um conjunto de 50 instâncias que se transformam em 150 instâncias variando a taxa de aprendizado $\alpha \in \{0,1; 0,2; 0,3\}$. Propomos duas heurísticas construtivas baseadas em regras de despacho. Também introduzimos uma busca local com diversas estratégias e vizinhanças para melhorar as soluções das heurísticas iniciais. Para a busca local, mostramos que, na presença do efeito de aprendizado, a abordagem clássica de considerar realocações de operações apenas no caminho crítico falha em considerar vizinhos potencialmente melhores do que a solução atual. Consequentemente, propusemos uma nova redução de vizinhança que não elimina vizinhos potencialmente melhores do que a solução atual e reduz a vizinhança em aproximadamente 50%. Além disso, propusemos um corte de vizinhança que reduz significativamente o tamanho da mesma (em cerca de uma ordem de magnitude) e encontra soluções que são no máximo 1% piores. A busca local introduzida e/ou as vizinhanças podem ser utilizados no desenvolvimento de meta-heurísticas de trajetória. Como trabalhos futuros, pretendemos considerar diferentes efeitos de aprendizado, que não dependem apenas da posição da operação na máquina à qual foi atribuída. Também pretendemos considerar funções objetivo que levem em conta o consumo de energia, e adaptar o problema para diversas aplicações do mundo real.

Referências

Alvarez-Valdes, R., Fuertes, A., Tamarit, J. M., Giménez, G., e Ramos, R. (2005). A heuristic to schedule flexible job-shop in a glass factory. *European Journal of Operational Research*, 165(2): 525–534.

Andrade-Pineda, J. L., Canca, D., Gonzalez-R, P. L., e Calle, M. (2020). Scheduling a dual-resource flexible job shop with makespan and due date-related criteria. *Annals of Operations Research*, 291(1):5–35.

Birgin, E. G., Feofiloff, P., Fernandes, C. G., De Melo, E. L., Oshiro, M. T. I., e Ronconi, D. P. (2014). A MILP model for an extended version of the flexible job shop problem. *Optimization Letters*, 8(4):1417–1431.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2022). *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 4th edition.

De Moerloose, P. e Maenhout, B. (2023). A two-stage local search heuristic for solving the steelmaking continuous casting scheduling problem with dual shared-resource and blocking constraints. *Operational Research*, 23(1):2.

Gan, P. Y. e Lee, K. S. (2002). Scheduling of flexible-sequenced process plans in a mould manufacturing shop. *The International Journal of Advanced Manufacturing Technology*, 20(3):214–222.

Leung, J. Y.-T., Li, H., e Pinedo, M. (2005). Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling*, 8(5):355–386.

Lunardi, W. T., Birgin, E. G., Laborie, P., Ronconi, D. P., e Voos, H. (2020). Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. *Computers and Operations Research*, 123:105020.

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br

Lunardi, W. T., Birgin, E. G., Ronconi, D. P., e Voos, H. (2021). Metaheuristics for the online printing shop scheduling problem. *European Journal of Operational Research*, 293(2):419–441.

Mastrolilli, M. e Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

Yu, L., Zhu, C., Shi, J., e Zhang, W. (2017). An extended flexible job shop scheduling model for flight deck scheduling with priority, parallel operations, and sequence flexibility. *Scientific Programming*, 2017:1–15.

DOI: 10.59254/sbpo-2024-193581
https://proceedings.science/p/193581?lang=pt-br