

Uma análise comparativa de modelos de Programação Linear Inteira Mista para o problema de programação de tarefas no ambiente jobshop flexível com restrições de blocking

Hanna Pamplona Hortencio

Departamento de Engenharia de Produção, Escola Politécnica, Universidade de São Paulo
Av. Prof. Luciano Gualberto, 1380, Cidade Universitária, 05508-010 São Paulo, SP, Brasil
hannapamplona@gmail.com

Débora Pretti Ronconi

Departamento de Engenharia de Produção, Escola Politécnica, Universidade de São Paulo
Av. Prof. Luciano Gualberto, 1380, Cidade Universitária, 05508-010 São Paulo, SP, Brasil
dronconi@usp.br

RESUMO

O problema de programação de tarefas no ambiente *jobshop* flexível com *blocking* (FBJSP) é um problema combinatório NP-difícil extensão do problema *jobshop* que incorpora flexibilidade e restrições de *blocking*. O FBJSP é relevante em diversos setores como a fabricação de semicondutores e centros cirúrgicos hospitalares. Este estudo desenvolve e compara dois modelos de Programação Linear Inteira Mista (PLIM) para o FBJSP, com o objetivo de minimizar o atraso total. O primeiro modelo é uma adaptação de Birgin et al. [2014] que foi selecionado por seu comprovado bom desempenho. O segundo modelo é uma adaptação de Dauzère-Pérès et al. [2024] que representa um desenvolvimento mais recente. Experimentos numéricos foram realizados com 80 instâncias adaptadas da literatura. Foi realizada uma extensa análise dos resultados obtidos por ambos os modelos.

PALAVRAS CHAVE. Problema de programação de tarefas no ambiente *jobshop* flexível, *blocking*, programação linear inteira mista

Tópicos OD, AD&GP, POI

ABSTRACT

The Flexible Blocking Job Shop Scheduling Problem (FBJSP) is an NP-hard combinatorial problem that extends the traditional job shop scheduling problem by incorporating machine flexibility and blocking constraints. The FBJSP is relevant in many sectors such as semiconductor manufacturing and hospital surgical centers. This study develops and compares two Mixed-Integer Linear Programming (MILP) models for the FBJSP, aiming to minimize total tardiness. The first model is an adaptation of Birgin et al. [2014], which was selected for its proven good performance. The second model is an adaptation of Dauzère-Pérès et al. [2024], which represents a more recent development. Numerical experiments were conducted using 80 instances adapted from the literature. An extensive analysis of the results obtained from both models was carried out.

KEYWORDS. Flexible jobshop scheduling problem, Blocking constraint, Mixed integer linear programming.

Paper topics OD, AD&GP, POI

1. Introdução

O Problema de Programação de Tarefas no ambiente *Jobshop* Flexível (FJSP, na sigla em inglês) é um problema combinatório NP-difícil amplamente aplicado em diferentes tipos de empresas, por exemplo: semicondutores, montagem de produtos, componentes aeronáuticos, fabricação de moldes, indústrias farmacêuticas e centros cirúrgicos hospitalares [Dauzère-Pérès et al., 2024; Pei et al., 2022; Pham e Klinkert, 2008]. Os ambientes de produção tornaram-se mais personalizados e complexos, exigindo a inclusão de novas restrições e funções objetivo, como capacidade limitada de estoques intermediários e datas de entrega, para melhor refletir a realidade industrial [Xie et al., 2019].

Em ambientes de produção é comum encontrarmos limitações de espaço para estoque (*buffer*) de produtos em processamento, chamamos essa restrição de *buffer* limitado. Um caso especial é o *buffer* zero, encontrado em processos que não permitem armazenamento de produtos em processamento, como hospitais [Pham e Klinkert, 2008]. Após a conclusão de uma operação, se não houver espaço disponível no *buffer*, a operação não pode continuar seu fluxo. Uma alternativa é manter a operação na máquina, o que impede o processamento de outras tarefas. Esse efeito na máquina é chamado de *blocking* [Groflin et al., 2011], e o problema abordado é denominado Problema de Programação de Tarefas no ambiente *Jobshop* Flexível com *Blocking* (FBJSP).

Embora o *makespan* seja a função objetivo mais comum na literatura sobre FJSP [Chaudhry e Khan, 2016], esse critério nem sempre reflete as reais necessidades das empresas. Para Sobeyko e Monch [2016], os critérios relacionados à data de entrega trazem benefícios para a competitividade das organizações. Segundo Mousakhani [2013], o atraso total total é a função objetivo mais comum relacionada à data de entrega.

Este trabalho aborda o FBJSP com *buffer* zero e atraso total como função objetivo. O objetivo deste estudo é desenvolver um modelo de Programação Linear Inteira Mista (PLIM) para esse problema e realizar uma análise comparativa dos resultados com outro modelo da literatura. Serão utilizadas instâncias conhecidas na literatura.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta uma revisão da literatura relacionada ao FBJSP. A Seção 3 detalha as adaptações desenvolvidas com adição de restrições de *buffer* zero e *blocking* e função objetivo de atraso total. A Seção 4 apresenta os resultados e análises, seguidos pelas conclusões na Seção 5.

2. Revisão de Literatura

O problema de programação de tarefas tem como objetivo alocar um número limitado de tarefas em máquinas sob algumas restrições. Cada tarefa é composta por várias operações que devem seguir uma sequência [Liang et al., 2022]. Um dos problemas de programação mais desafiadores ocorre no ambiente *Jobshop* Flexível. O Problema de Programação de Tarefas em ambiente *Jobshop* Flexível (FJSP, na sigla em inglês) é uma generalização do problema clássico de programação de tarefas no ambiente *jobshop* (JSP), no qual cada tarefa segue uma rota específica pelas máquinas [Dauzère-Pérès et al., 2024].

O FJSP pode possuir três tipos de flexibilidade: sequenciamento, operacional e processamento. Na flexibilidade de sequenciamento, as operações de uma tarefa podem ter mais de uma ordem de sequenciamento. Na flexibilidade operacional, para cada operação existe um conjunto de máquinas capazes de processá-la. E na flexibilidade de processamento, cada tarefa possui algumas rotas alternativas para escolher [Dauzère-Pérès et al., 2024]. Embora aumente a complexidade das decisões, a flexibilidade torna o problema mais fiel à realidade de cenários do mundo real [Dauzère-Pérès et al., 2024].

Este trabalho aborda a flexibilidade operacional, em que cada operação pode ser processada por uma entre várias máquinas elegíveis. Em alguns problemas, máquinas com funções

semelhantes podem ser agrupadas em centros de trabalho [Poppenborg et al., 2012; Gomes et al., 2013]. No entanto, em problemas de programação que envolvem máquinas multipropósito, organizar as máquinas em centros de trabalho não é viável. Este estudo considera o caso mais geral, onde os centros de trabalho não são utilizados.

Dada a complexidade do problema, muitos estudos assumem *buffer* ilimitado entre as máquinas. No entanto, ambientes reais de manufatura e de serviço frequentemente enfrentam limitações de espaço para o estoque de produtos em processamento. Essa restrição é conhecida como *buffer* limitado, ou *buffer* zero quando não é permitido qualquer estoque entre as máquinas [Xie et al., 2019; Gao et al., 2020].

O *blocking* ocorre quando há *buffer* zero ou os *buffers* estão cheios, fazendo com que a tarefa continue ocupando a máquina anterior até que haja espaço para avançar. Máquinas em *blocking* não podem processar outras tarefas [Teymourifar et al., 2020; McCormick et al., 1989]. Na mais recente revisão da literatura sobre o FJSP, o *blocking* foi identificado como uma restrição desafiadora, pois pode levar a muitas soluções inviáveis [Dauzère-Pérès et al., 2024].

Segundo Sobeyko e Monch [2016], os critérios relacionados à data de entrega trazem benefícios para a satisfação do cliente e para a competitividade, embora a maioria dos estudos na literatura não enfatize esse objetivo. Dauzère-Pérès et al. [2024] descrevem três tipos de objetivos relacionados ao atraso: atraso máximo (para evitar atrasos extremos), atraso total (focado apenas no tempo de atraso) e atraso total ponderado (que também depende do cliente). Embora o atraso total ponderado seja uma opção amplamente considerada na indústria, a definição dos pesos não é uma tarefa trivial, motivo pelo qual esse estudo adota o atraso total como função objetivo.

A seguir, será apresentada uma breve revisão literatura com os trabalhos mais relevantes sobre FBJSP.

Diversos trabalhos apresentam métodos não exatos para resolver esse problema, como regras de sequenciamento [Teymourifar et al., 2020], heurísticas de busca local em duas etapas [Moerlose e Maenhout, 2023], heurística gulosa [Mati e Xie, 2011], busca tabu [Groflin et al., 2011], abordagem genética [Mati et al., 2011] e meta-heurísticas híbridas [Burdett e Kozan, 2017]. Alguns desses estudos também incorporam restrições adicionais, como tempos de transferência e de *setup*.

Até onde sabemos, apenas alguns trabalhos propõe formulações de Programação Linear Inteira Mista (do inglês, MILP) para o FBJSP. Estes trabalhos serão descritos a seguir.

Em Pham e Klinkert [2008], o processo de programação de pacientes em um centro cirúrgico é apresentado como um FBJSP. Um MILP é desenvolvido e testado em um pequeno conjunto de instâncias. Uma etapa de pré-processamento é necessária, na qual os recursos necessários para realizar uma cirurgia são agrupados em módulos. Essa estratégia reduz a complexidade da alocação de recursos. O objetivo é minimizar a soma dos tempos de início de todas as operações, buscando tornar o sistema disponível o mais cedo possível.

Em Poppenborg et al. [2012], uma Busca Tabu é usada para resolver o FBJSP com restrições de transporte. A função objetivo visa minimizar o atraso total ponderado. Os autores utilizam um modelo MILP para comparar os resultados heurísticos em instâncias de pequeno porte. Em seu modelo MILP, as máquinas são organizadas em grupos com base nas funções que desempenham, e as operações são atribuídas à esses grupos. Além disso, cenários online são testados, nos quais as tarefas chegam enquanto a programação já está em andamento.

Em Gomes et al. [2013], um processo de produção sob encomenda (*make-to-order*) é descrito por meio de um modelo MILP com *buffers* intermediários limitados. Esse trabalho também inclui restrições como fluxo reentrante e operações de montagem (*assembly operations*). Nesse problema, as tarefas são atribuídas a rotas pré-definidas e as máquinas são organizadas em grupos

de máquinas paralelas não idênticas. A função multiobjetivo considera a soma ponderada do atraso, adiantamento e estoque em processo. Os autores também apresentam um algoritmo para inserir novas tarefas em uma programação já em andamento. Um pequeno conjunto de instâncias foi gerado com base em dados de uma indústria de fabricação de moldes.

O artigo de Dauzère-Pérès et al. [2024] é uma revisão da literatura que analisa e classifica pesquisas dos últimos trinta anos sobre o FJSP. A revisão organiza os artigos de acordo com a função objetivo, as restrições e os métodos de solução empregados. Os autores também propõem um modelo MILP genérico com minimização do *makespan* e uma restrição adicional de *blocking*.

O objetivo deste trabalho é propor um modelo MILP para o FBJSP com *buffer* zero e minimização do atraso total. Para atingir isso, foi feita uma adaptação do modelo de Birgin et al. [2014], que apresenta uma formulação MILP concisa para o FJSP minimização do *makespan*. Esse modelo foi avaliado e apresenta desempenho superior ao modelo de Ozguven et al. [2010].

Nesse trabalho, também foram realizados testes numéricos para comparar o modelo proposto com o mais recente desenvolvido por Dauzère-Pérès et al. [2024], que inclui uma restrição de *blocking*. Ajustes foram feitos no modelo para considerar o objetivo de minimizar o atraso total.

3. Modelos adaptados

Esta seção descreve as adaptações feitas nos modelos propostos por Birgin et al. [2014] e Dauzère-Pérès et al. [2024]. No primeiro, o objetivo de minimizar o atraso total e restrições de *blocking* com *buffer* zero foram adicionados; no segundo, o objetivo foi ajustado para minimizar o atraso total.

3.1. Adaptação do modelo Birgin et al. [2014]

O modelo de Birgin et al. [2014] é um modelo compacto quando comparado a outros modelos para o FJSP. Isso ocorre em parte porque o modelo considera as operações de forma independente e estabelece sua precedência utilizando o conjunto A . Nesta adaptação, denominada $Birgin_{Block}^{TT}$, restrições de *buffer* zero e de *blocking* são incorporadas, e a função objetivo é modificada para minimizar o atraso. A Tabela 1 apresenta a notação dos autores e as adições feitas na adaptação do modelo original. O modelo $Birgin_{Block}^{TT}$ é apresentado a seguir.

$$\min \sum_{i \in O_f} T_i \quad (1)$$

Sujeito a

$$\sum_{k \in F_i} x_{ik} = 1 \quad \forall i \in O \quad (2)$$

$$p'_i = \sum_{k \in F_i} x_{ik} p_{ik} \quad \forall i \in O \quad (3)$$

$$y_{ij} + y_{ji} \geq x_{ik} + x_{jk} - 1 \quad \forall k \in M \quad \forall (i, j) \in B_k \quad (4)$$

$$D_i + p'_j - (1 - y_{ij})L \leq D_j \quad \forall (i, j) \in B \quad (5)$$

$$D_i + p'_j \leq D_j \quad \forall (i, j) \in A \quad (6)$$

$$D_i \geq D_j - (3 - y_{jh} - x_{jk} - x_{hk})L \quad \forall (j, h) \in B_k \quad \forall (i, h) \in A \quad \forall k \in M \quad i \neq 0 \quad (7)$$

$$T_i \geq D_i - dd_i \quad \forall i \in O_f \quad (8)$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in O \quad \forall k \in M \quad (9)$$

$$y_{ij} \in \{0, 1\} \quad \forall (i, j) \in B \quad (10)$$

$$T_i \geq 0 \quad \forall i \in O_f \quad (11)$$

$$D_i \geq 0 \quad \forall i \in O \quad (12)$$

Tabela 1: Notações do modelo $Birgin_{Block}^{TT}$

Índices e conjuntos

M	conjunto de máquinas, $k \in M$
O	conjunto de operações, $i, j, h \in O$; $i = 0$ denota uma operação artificial
A	conjunto de arcos (i, j) que representam restrições de precedência entre operações
F_i	subconjunto de máquinas que podem processar a operação i , $F_i \neq \emptyset$
O_k	subconjunto de operações i que podem ser processadas na máquina k
B_k	conjunto de todos os pares ordenados (i, j) de elementos distintos de O_k , $i \neq j$
B	conjunto de todos os pares (i, j) em B_k , $B = \bigcup_k B_k$ apenas se $i \neq j$ e $F_i \cap F_j \neq \emptyset$
O_f	subconjunto com a última operação de cada tarefa, $O_f \subset O$ (notação adicional)

Parâmetros

p_{ik}	tempo de processamento da operação i na máquina k
L	número grande, $L = \sum_{i \in O} \max_{k \in F_i} p_{ik}$
dd_i	data de entrega da operação i , $i \in O_f$ (notação adicional)

Variáveis de decisão

x_{ik}	1, se a operação i for atribuída à máquina k ; 0, caso contrário
y_{ij}	1, se a operação i for programada antes da operação j , $(i, j) \in B$; 0, caso contrário
p'_i	tempo de processamento da operação i na máquina atribuída
T_i	atraso da operação i , $i \in O_f$ (notação adicional)
D_i	instante de partida da operação i , $i \in O$; $D_0 = 0$ (notação adicional)

A função objetivo (1) minimiza o atraso total. O primeiro conjunto de restrições (2) garante que cada operação seja atribuída a uma máquina capaz de processá-la. Nas restrições (3), define-se o tempo de processamento na máquina atribuída. As restrições (4), quando ativas ($x_{ik} = x_{jk} = 1$), determinam que deve haver uma ordem entre as operações processadas na mesma máquina. Nas restrições (5) essa ordem é definida, juntamente com os tempos de partida das operações. Os conjuntos de restrições (4) e (5) garantem, em conjunto, a precedência entre operações que utilizam a mesma máquina. O conjunto (6) define as restrições de precedência entre duas operações de uma mesma tarefa e seus respectivos tempos de partida.

As restrições (7) garantem o *blocking* com *buffer* zero. A Figura 1 ilustra sua funcionalidade utilizando três operações $i, j, h \in O$. O par $(i, h) \in A$ representa operações consecutivas de uma mesma tarefa, e h e j compartilham a máquina M1. O *blocking* ocorre após i (do instante 5 ao 7), com a máquina M2 bloqueada até que a operação h comece na máquina M1. Para que isso ocorra, é necessário garantir que a operação j finalize seu processamento simultaneamente ou antes da operação h . O conjunto de restrições (8) calcula o atraso da última operação de cada tarefa. Por fim, as restrições (9)-(12) definem o domínio das variáveis de decisão.

3.2. Adaptação do modelo de Dautère-Pérès et al. [2024]

O modelo de Dautère-Pérès et al. [2024] também desconsidera a necessidade de um conjunto de tarefas e utiliza apenas o conjunto de operações. A precedência entre operações é definida pelos parâmetros $pr(i)$ e $fr(i)$, que representam, respectivamente, o predecessor e o sucessor da operação i . No modelo adaptado, denominado $DPeres^{TT}$, a função objetivo foi modificada para a minimização do atraso. As notações estão resumidas na Tabela 2, incluindo aquelas necessárias para a adaptação do modelo. O modelo $DPeres^{TT}$ é apresentado a seguir.

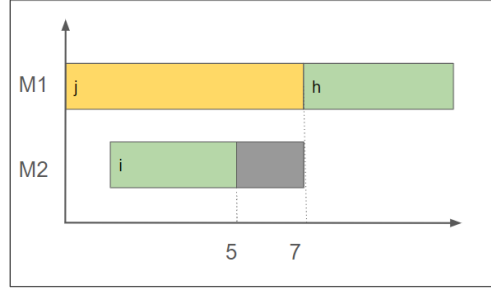


Figura 1: Exemplo com *buffer zero* e restrição de *blocking*.

Tabela 2: Notações do modelo DP_{Peres}^{TT}

Índice e conjuntos

R	conjunto de máquinas, $k \in R$
O	conjunto de operações, $i, j \in O$
R_i	subconjunto de máquinas que podem processar a operação i , $R_i \subseteq R$
O_f	subconjunto com a última operação de cada tarefa, $O_f \subset O$ (notação adicional)

Parâmetros

$pr(i)$	operação que precede imediatamente a operação i
$fr(i)$	operação que sucede imediatamente a operação i
p_i^k	tempo de processamento da operação i na máquina k
H	valor suficientemente grande, $H = \sum_{i \in O} \sum_{k \in R_i} p_i^k$
dd_i	data de entrega da operação i , $i \in O_f$ (notação adicional)

Variáveis de decisão

α_i^k	1, se a operação i for atribuída à máquina k ; 0, caso contrário
β_{ij}	1, se a operação i for executada antes da operação j ; 0, caso contrário
t_i	tempo de início da operação i
T_i	atraso da operação i , $i \in O_f$ (notação adicional)
C_i	tempo de conclusão da operação i , $i \in O_f$ (notação adicional)

$$\min \sum_{i \in O_f} T_i \quad (13)$$

Sujeito a

$$\sum_{k \in R_i} \alpha_i^k = 1 \quad \forall i \in O \quad (14)$$

$$t_i \geq t_{pr(i)} + \sum_{k \in R_{pr(i)}} p_{pr(i)}^k \alpha_{pr(i)}^k \quad \forall i \in O \quad (15)$$

$$t_i \geq t_{fr(j)} - (2 - \alpha_i^k - \alpha_j^k + \beta_{ij})H \quad \forall (i, j) \in O \times O, i \neq j \neq fr(j), j \notin O_f, k \in R_i \cap R_j \quad (16)$$

$$t_i \geq t_j + p_j^k - (2 - \alpha_i^k - \alpha_j^k + \beta_{ij})H \quad \forall (i, j) \in O \times O, j \in O_f, i \neq j, k \in R_i \cap R_j \quad (17)$$

$$t_j \geq t_{fr(i)} - (3 - \alpha_i^k - \alpha_j^k - \beta_{ij})H \quad \forall (i, j) \in O \times O, i \neq j \neq fr(i), j \notin O_f, k \in R_i \cap R_j \quad (18)$$

$$t_j \geq t_i + p_i^k - (3 - \alpha_i^k - \alpha_j^k - \beta_{ij})H \quad \forall (i, j) \in O \times O, i \in O_f, i \neq j, k \in R_i \cap R_j \quad (19)$$

$$C_i \geq t_i + \sum_{k \in R_i} p_i^k \alpha_i^k \quad \forall i \in O_f \quad (20)$$

$$T_i \geq C_i - dd_i \quad \forall i \in O_f \quad (21)$$

$$\alpha_i^k \in \{0, 1\} \quad \forall i \in O, k \in R_i \quad (22)$$

$$\beta_{ij} \in \{0, 1\} \quad \forall (i, j) \in O \times O \quad (23)$$

$$T_i, C_i \geq 0 \quad \forall i \in O_f \quad (24)$$

A função objetivo (13) minimiza o atraso total para todas as operações finais. O conjunto de restrições (14) garante que cada operação seja processada por apenas uma máquina. As restrições (15) definem os tempos de início das operações, considerando a precedência entre as operações de uma mesma tarefa. As restrições (16) e (18) garantem, em conjunto, que as operações atribuídas à mesma máquina não se sobreponham e que a restrição de *blocking* seja satisfeita. Dois conjuntos adicionais de restrições (17) e (19) foram introduzidos para possibilitar o cálculo correto dos tempos de início das operações finais de cada tarefa, o que viabiliza o cálculo da nova função objetivo. O conjunto de restrições (20) calcula o tempo de conclusão das operações finais de cada tarefa, enquanto as restrições (21) definem o atraso dessas operações. Por fim, as restrições (22)-(24) definem o domínio das variáveis de decisão.

4. Experimentos numéricos

Nesta seção serão apresentados os resultados dos experimentos numéricos. Os modelos foram implementados no Gurobi (gurobi-11.0.3-py312.0) com limite de tempo de CPU definido para uma hora. Os experimentos foram realizados no cluster de uma universidade brasileira com processador Intel (R) Xeon (R) CPU E7-2870, 20 núcleos, 512 GB de RAM, 2.40 GHz e Linux 7.5 como sistema operacional.

Foram utilizados dois conjuntos de instâncias comumente empregados na literatura: o Set 1 (WT1-WT5), desenvolvido por Brandimarte [1993], e o Set 2, uma adaptação de Sobeyko e Monch [2016] baseada nas instâncias (Mk01-Mk10) de Brandimarte [1993], com datas de entrega ajustadas para criar as instâncias Apertadas (Tight), Moderadas (Moderated) e Folgadas (Loose). Estes conjuntos são referidos como *Set1_original* e *Set2_original*, respectivamente. Para contornar a dificuldade de resolver grandes instâncias em um tempo razoável, foi criado um conjunto menor de instâncias. No caso do Set 1, o número de operações foi reduzido para aproximadamente 30%, 50% e 80% das quantidades originais, enquanto o Set 2 teve uma redução aproximada de 50%. No total, 80 instâncias foram avaliadas. A Tabela 3 mostra o número de operações e máquinas para cada instância.

De acordo com a Teoria da Complexidade [Cook, 1971], a quantidade de recursos, tempo e memória necessários para processar um problema estão intimamente relacionados ao número de variáveis, principalmente inteiras ou binárias, e restrições. Uma análise foi conduzida e os resultados apontam que o modelo DP_{Peres}^{TT} possui um número menor de variáveis e restrições do que o modelo $Birgin_{Block}^{TT}$ na maioria das instâncias, com variações no número de variáveis binárias entre 0,29% a 12,00% e no número de restrições entre 3,10% a 27,41%.

As Tabelas 4 e 5 apresentam os resultados dos testes numéricos. Em 22 das 80 instâncias testadas, nenhum dos dois modelos foi capaz de encontrar uma solução factível. Portanto, essas

Tabela 3: Número de operações e máquinas para cada instância

Set 1	WT1	WT2	WT3	WT4	WT5					
# Op. Original	50	100	239	242	354					
# Op. 80%	40	80	194	193	279					
# Op. 50%	25	50	126	121	181					
# Op. 30%	15	30	74	73	105					
# Máquinas	5	5	10	10	15					
Set 2	Mk01	Mk02	Mk03	Mk04	Mk05	Mk06	Mk07	Mk08	Mk09	Mk10
# Op. Original	55	58	150	90	106	150	100	225	238	240
# Op. 50%	27	30	80	53	54	80	50	116	120	122
# Máquina	6	6	8	8	4	10	5	9	10	11

instâncias não serão incluídas na tabela. Para cada instância e modelo, os resultados são apresentados em duas colunas. A coluna *Atraso total* indica o valor da função objetivo, quando a solução ótima é encontrada. Caso contrário, a coluna apresenta os valores do melhor limite inferior, superior e o GAP (25). O tempo computacional necessário para atingir esse resultado é apresentado na coluna *Tempo CPU (s)* e foi limitado a uma hora (3600 segundos).

$$GAP = \frac{Upper_bound - Lower_bound}{Upper_bound} \quad (25)$$

As duas adaptações se mostraram adequadas para representar o problema proposto, com a solução ótima sendo encontrada em 27 instâncias (34%) considerando a execução de ambos os modelos. Dentre essas, apenas duas instâncias (Set1_WT1_80% e Set1_WT4_80%) tiveram a solução ótima encontrada com a execução de apenas um dos modelos, nas demais 25 instâncias, a execução de ambos os modelos levou à solução ótima. Destaca-se a instância WT1_80%, na qual a execução do modelo $Birgin_{Block}^{TT}$ encontrou e comprovou a solução ótima. Já na execução do modelo $DPeres^{TT}$, embora a mesma solução tenha sido encontrada (com upper bound igual a 23), não foi possível comprovar sua otimalidade.

Em cinco instâncias, uma solução factível foi obtida apenas com a execução de um dos modelos, enquanto o outro não foi capaz de encontrar uma solução factível. Em quatro dessas instâncias, o modelo $DPeres^{TT}$ apresentou desempenho superior, enquanto em uma instância, o modelo $Birgin_{Block}^{TT}$ obteve melhor desempenho.

Analisou-se o GAP nas 26 instâncias em que ambas as execuções encontraram uma solução factível. Nessas, a execução do modelo $Birgin_{Block}^{TT}$ apresentou melhor desempenho em 11 casos, enquanto a execução do modelo $DPeres^{TT}$ teve melhor desempenho em 8. Nas 7 instâncias restantes, em que ambas as execuções resultaram em GAP de 100%, foram comparados os limites superiores. Nesse caso, a execução do modelo $Birgin_{Block}^{TT}$ foi superior em 3 instâncias e a do $DPeres^{TT}$ em 4.

Como esperado, a execução dos modelos não encontrou a solução ótima para as instâncias originais. No Set 1, a execução dos modelos encontrou a solução ótima para todas as instâncias com operações reduzidas de 50% e 30%. No Set 2, a execução dos modelos encontrou a solução ótima em 9 das 10 instâncias com prazos de entrega folgados. Para prazos de entrega moderados e apertados, apenas 3 de 10 instâncias encontraram o resultado ótimo.

Quanto ao tempo de processamento, os modelos apresentaram desempenho bastante si-

Tabela 4: Solução para cada modelo e instância - parte 1

Instância	Atraso total		Tempo CPU (s)	
	$Birgin_{Block}^{TT}$	$DPeres^{TT}$	$Birgin_{Block}^{TT}$	$DPeres^{TT}$
Set1_WT1	[5,120] 96%	[0,133] 100%	3600.0	3600.0
Set1_WT1_80%	23	[4,23] 83%	1637.0	3600.0
Set1_WT1_50%	0	0	1.0	0.0
Set1_WT1_30%	0	0	1.0	0.0
Set1_WT2	[20,480] 96%	[21,615] 97%	3600.0	3600.0
Set1_WT2_80%	[21,198] 89%	[22,223] 90%	3600.0	3600.0
Set1_WT2_50%	2	2	45.0	14.0
Set1_WT2_30%	1	1	0.0	0.0
Set1_WT3_50%	0	0	102.0	141.0
Set1_WT3_30%	0	0	2.0	1.0
Set1_WT4_80%	-	54	3600.0	856.0
Set1_WT4_50%	51	51	27.0	9.0
Set1_WT4_30%	0	0	1.0	0.0
Set1_WT5_50%	0	0	2322.0	445.0
Set1_WT5_30%	0	0	3.0	4.0
Set2_Mk01_Loose	[21,57] 63%	[22,48] 54%	3600.0	3600.0
Set2_Mk01_Moderate	[68,122] 44%	[73,122] 40%	3601.0	3600.0
Set2_Mk01_Tight	[153,198] 22%	[146,212] 31%	3601.0	3600.0
Set2_Mk01_Loose_50%	4	4	1.0	2.0
Set2_Mk01_Moderate_50%	30	30	7.0	2.0
Set2_Mk01_Tight_50%	73	73	4.0	3.0
Set2_Mk02_Loose	[0,45] 100%	[0,46] 100%	3600.0	3600.0
Set2_Mk02_Moderate	[25,101] 76%	[21,84] 75%	3600.0	3600.0
Set2_Mk02_Tight	[99,163] 40%	[97,162] 40%	3600.0	3600.0
Set2_Mk02_Loose_50%	0	0	1.0	2.0
Set2_Mk02_Moderate_50%	7	7	20.0	6.0
Set2_Mk02_Tight_50%	44	44	8.0	10.0
Set2_Mk03_Moderate	-	[96,1597] 94%	3600.0	3600.0
Set2_Mk03_Tight	-	[486,2259] 78%	3600.0	3600.0
Set2_Mk03_Loose_50%	0	0	9.0	10.0
Set2_Mk03_Moderate_50%	[55,72] 24%	[55,78] 29%	3600.0	3600.0
Set2_Mk03_Tight_50%	[293,359] 18%	[295,328] 10%	3601.0	3600.0
Set2_Mk04_Loose	[36,199] 82%	[13,186] 93%	3600.0	3600.0
Set2_Mk04_Moderate	[130,298] 56%	[94,353] 73%	3600.0	3600.0
Set2_Mk04_Tight	[307,497] 38%	[272,522] 48%	3600.0	3600.0
Set2_Mk04_Loose_50%	2	2	16.0	14.0
Set2_Mk04_Moderate_50%	70	70	3051.0	334.0
Set2_Mk04_Tight_50%	168	168	1513.0	2848.0

milar na maioria das instâncias. Entretanto, destacam-se algumas instâncias com diferenças significativas nos tempos de execução. Por exemplo, nas instâncias WT5_50% e Mk04_Moderate_50%, a execução do modelo $DPeres^{TT}$ atingiu a solução ótima com diferença de 31 e 45 minutos, respectivamente. Em contrapartida, na instância Mk04_Tight_50%, a execução do modelo $Birgin_{Block}^{TT}$

Tabela 5: Solução para cada modelo e instância - parte 2

Instância	Atraso total		Tempo CPU (s)	
	$Birgin_{Block}^{TT}$	$DPeres^{TT}$	$Birgin_{Block}^{TT}$	$DPeres^{TT}$
Set2_Mk05_Loose_50%	0	0	7.0	1.0
Set2_Mk05_Moderate_50%	[0,80] 100%	[0,66] 100%	3600.0	3600.0
Set2_Mk05_Tight_50%	[134,305] 56%	[152,322] 53%	3600.0	3600.0
Set2_Mk06_Loose_50%	0	0	2.0	1.0
Set2_Mk06_Moderate_50%	[2,7] 71%	[0,7] 100%	3600.0	3600.0
Set2_Mk06_Tight_50%	[91,111] 18%	[86,110] 21%	3600.0	3600.0
Set2_Mk07_Loose	-	[0,619] 100%	3600.0	3600.0
Set2_Mk07_Moderate	[0,869] 100%	-	3601.0	3600.0
Set2_Mk07_Loose_50%	0	0	4.0	0.0
Set2_Mk07_Moderate_50%	[0,21] 100%	[0,12] 100%	3600.0	3600.0
Set2_Mk07_Tight_50%	[56,231] 76%	[66,233] 72%	3600.0	3600.0
Set2_Mk08_Loose_50%	0	0	178.0	575.0
Set2_Mk08_Moderate_50%	[0,443] 100%	[0,534] 100%	3600.0	3601.0
Set2_Mk08_Tight_50%	[500,1450] 66%	[583,1346] 57%	3600.0	3600.0
Set2_Mk09_Loose_50%	[0,65] 100%	[0,32] 100%	3601.0	3600.0
Set2_Mk09_Moderate_50%	[169,416] 59%	[153,474] 68%	3600.0	3600.0
Set2_Mk09_Tight_50%	[754,985] 23%	[799,970] 18%	3600.0	3600.0
Set2_Mk10_Loose_50%	0	0	374.0	156.0
Set2_Mk10_Moderate_50%	[0,248] 100%	[0,405] 100%	3600.0	3600.0
Set2_Mk10_Tight_50%	-	[434,919] 53%	3600.0	3600.0

superou o $DPeres^{TT}$, alcançando a solução ótima 22 minutos antes.

5. Conclusões

Este artigo aborda o problema de programação de tarefas no ambiente *Jobshop* Flexível com *blocking* (FBJSP), zero *buffer* e com objetivo de minimização do atraso total. Foi realizada uma breve revisão da literatura com o intuito de identificar estudos que tratam desse problema. Quatro artigos que apresentam modelos MILP para o FBJSP foram identificados e descritos.

O objetivo deste trabalho foi desenvolver um novo modelo MILP para o FBJSP com zero *buffer* e atraso total como função objetivo. Para alcançar esse objetivo, adaptamos o modelo MILP apresentado em Birgin et al. [2014], que apresenta resultados superiores a outro modelo conhecido na literatura. Este trabalho também realizou experimentos numéricos e comparou os resultados com um modelo recente adaptado da literatura, proposto por Dauzère-Pérès et al. [2024].

Ambos modelos desenvolvidos nesse trabalho demonstraram-se apropriados para descrever adequadamente o problema analisado. A execução de cada modelo resultou na obtenção da solução ótima em 26 das 80 (32%) instâncias analisadas. Considerando a execução dos dois modelos, aproximadamente 72% das instâncias testadas encontraram uma solução viável. A análise dos tempos computacionais indica que os modelos apresentaram resultados semelhantes, com apenas três instâncias mostrando durações significativamente diferentes, variando de 22 a 45 minutos.

A solução ótima não foi encontrada para as instâncias originais, o que era esperado. No entanto, a solução ótima foi alcançada para todas as instâncias com operações reduzidas de 50% e 30% do Set 1. No Set 2, a solução ótima foi encontrada em 9 de 10 instâncias com prazos folgados, indicando que prazos mais flexíveis facilitam a obtenção da solução ótima.

Vale ressaltar que os dois modelos desenvolvidos são capazes, com pequenas adaptações, de processar instâncias com rotas não lineares (*non-linear routes*). Rotas não lineares são fenômenos observados em diversas indústrias envolvendo processos de montagem ou desmontagem. Em processos de montagem, duas ou mais operações independentes e simultâneas são seguidas por uma única operação subsequente. Por outro lado, em processos de desmontagem, uma operação se divide em várias operações que seguem de forma independente e simultânea. Realizamos alguns testes utilizando as instâncias geradas em Birgin et al. [2014].

Para adaptar o modelo $Birgin_{Block}^{TT}$, é necessário adicionar um conjunto de restrições (26) que garante que todas as operações predecessoras de uma operação comecem simultaneamente.

$$D_i = D_j \quad \forall i, j, k \in O : (i, k) \in A \wedge (j, k) \in A \quad (26)$$

Para adaptar o modelo $DPeres^{TT}$, os parâmetros $pr(i)$ e $fr(i)$ devem ser substituídos por um conjunto de operações predecessoras e sucessoras de i , respectivamente. Com essas pequenas adaptações, ambos os modelos são capazes de processar rotas não lineares.

Como sugestão para trabalhos futuros, recomendamos testar os modelos com instâncias adicionais com rotas não lineares para determinar se há alguma diferença no desempenho dos dois modelos.

Referências

- 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*, 8:1417–1431.
- Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183.
- Burdett, R. L. e Kozan, E. (2017). An integrated approach for scheduling health care activities in a hospital. *European Journal of Operational Research*, 264:756–773.
- Chaudhry, I. A. e Khan, A. A. (2016). A research survey: Review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23:551–591.
- Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, p. 151–158, New York, NY, USA. Association for Computing Machinery.
- Dauzère-Pérès, S., Ding, J., Shen, L., e Tamssaouet, K. (2024). The flexible job shop scheduling problem: A review. ISSN 03772217.
- Gao, Z., Feng, Y., e Xing, K. (2020). A hybrid estimation-of-distribution algorithm for scheduling flexible job shop with limited buffers based on petri nets. *IEEE Access*, 8:165396–165408. ISSN 21693536.
- Gomes, Barbosa-Povoa, e Novais (2013). Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach. *International Journal of Production Research*, 51(17):5120–5141.
- Groflin, H., Pham, D. N., e Burgy, R. (2011). The flexible blocking job shop with transfer and set-up times. *Journal of Combinatorial Optimization*, 22:121–144. ISSN 13826905.

- Liang, Z., Zhong, P., Liu, M., Zhang, C., e Zhang, Z. (2022). A computational efficient optimization of flow shop scheduling problems. *Scientific Reports*, 12. ISSN 20452322.
- Mati e Xie (2011). Multiresource shop scheduling with resource flexibility and blocking. *IEEE Transactions on Automation Science and Engineering*, 8(1):175–189.
- Mati, Y., Lahlou, C., e Dauzere-Peres, S. (2011). Modelling and solving a practical flexible job-shop scheduling problem with blocking constraints. *International Journal of Production Research*, 49: 2169–2182. ISSN 00207543.
- 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*, 37:925–935. ISSN 0030364X.
- Moerlose, 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):1–43. URL https://ideas.repec.org/a/spr/operea/v23y2023i1d10.1007_s12351-023-00760-2.html.
- Mousakhani, M. (2013). Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. *International Journal of Production Research*, 51.
- Ozguven, C., Ozbakir, L., e Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34:1539–1548.
- Pei, F., Zhang, J., Mei, S., e Song, H. (2022). Critical review on the objective function of flexible job shop scheduling. ISSN 15635147.
- Pham, D. N. e Klinkert, A. (2008). Surgical case scheduling as a generalized job shop scheduling problem. *European Journal of Operational Research*, 185:1011–1025. ISSN 0377-2217.
- Poppenborg, J., Knust, S., e Hertzberg, J. (2012). Online scheduling of flexible job-shops with blocking and transportation.
- Sobeyko, O. e Monch, L. (2016). Heuristic approaches for scheduling jobs in large-scale flexible job shops. *Computers and Operations Research*, 68:97–109. ISSN 03050548.
- Teymourifar, A., Ozturk, G., Ozturk, Z. K., e Bahadir, O. (2020). Extracting new dispatching rules for multi-objective dynamic flexible job shop scheduling with limited buffer spaces. *Cognitive Computation*, 12:195–205. ISSN 18669964.
- Xie, J., Gao, L., Peng, K., Li, X., e Li, H. (2019). Review on flexible job shop scheduling. ISSN 25168398.