DANIEL LUPOVICI BORGER

PROGRAMAÇÃO DA PRODUÇÃO – MINIMIZAÇÃO DO ATRASO TOTAL NO AMBIENTE *FLOW SHOP* COM RESTRIÇÕES DE DISPONIBILIDADE

SÃO PAULO 2021

DANIEL LUPOVICI BORGER

PROGRAMAÇÃO DA PRODUÇÃO – MINIMIZAÇÃO DO ATRASO TOTAL NO AMBIENTE *FLOW SHOP* COM RESTRIÇÕES DE DISPONIBILIDADE

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de Engenheiro de Produção.

Orientadora: Prof. Dra. Débora Pretti Ronconi

SÃO PAULO 2021

Catalogação-na-publicação

Borger, Daniel Lupovici
PROGRAMAÇÃO DA PRODUÇÃO – MINIMIZAÇÃO DO ATRASO TOTAL NO AMBIENTE FLOW SHOP COM RESTRIÇÕES DE DISPONIBILIDADE / D. L. Borger -- São Paulo, 2021.
59 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção.

1.PESQUISA OPERACIONAL 2.PROGRAMAÇÃO DA PRODUÇÃO I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Produção II.t.

AGRADECIMENTOS

Aos meus pais Roberto e Eliane e ao meu irmão David pelo carinho e apoio durante todos os momentos.

Às minhas avós, aos meus primos e tios queridos pelos bons momentos em família que compartilhamos e por todo apoio.

Aos meus amigos da faculdade e à Equipe de Xadrez da Poli, pelo companheirismo e por tornarem essa jornada politécnica mais valiosa e prazerosa.

À Lara, pelo amor, parceria e pela motivação que eu precisei nos momentos difíceis.

À professora Dra. Débora Pretti Ronconi, por despertar o meu interesse na área de Pesquisa Operacional e por todo apoio, orientação e paciência neste projeto.

If you wait for luck to turn up,

life becomes very boring.

(Mikhail Tal)

RESUMO

Este trabalho teve como objeto de estudo o problema de minimização do atraso total em um *flow shop* permutacional de duas máquinas com uma janela de indisponibilidade *non-resumable* (que não permite um job ser interrompido pela indisponibilidade e retomado depois) em cada uma das máquinas, inédito na literatura. Foi criada uma formulação de Programação Linear Inteira Mista desse problema e uma implementação com o *solver Gurobi* para resolver o problema de maneira ótima.

Como o problema estudado neste trabalho é NP-Difícil, foram também propostos métodos heurísticos, sendo duas heurísticas construtivas baseadas no algoritmo NEH e uma regra de despacho EDD (*Earliest Due Date*). Esse uso de heurísticas é importante para problemas NP-Difíceis porque em geral instâncias grandes desses problemas não podem ser resolvidos por métodos exatos em um tempo razoável para a aplicação.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 Introdução

1.1 Contexto

O problema de Programação da Produção (*Scheduling*) tem papel importante na tomada de decisões relacionadas a alocação de recursos, sendo muito aplicado em sistemas produtivos, de manufatura e de processamento computacional (Pinedo 2012).

Melhores programas de produção (*schedules*) trazem uma vantagem competitiva para a empresa através de ganhos em produtividade dos recursos e outras eficiências relacionadas à gestão de operações, o que motiva a busca por abordagens mais eficazes de Programação da Produção (Rodammer e White Jr 1988).

Os problemas de Programação da Produção costumam ser categorizados pela configuração dos recursos no sistema produtivo (explicado na Seção 2.1 em mais detalhe). Este trabalho explora um problema na configuração de *flow shop*, layout muito comum na manufatura, em que *m* máquinas estão alocadas em série para o processamento das atividades (*jobs*) e todos os *jobs* devem seguir a mesma sequência de máquinas (Pinedo 2012).

Outro fator que define um problema de programação da produção é o critério de otimização. O problema estudado neste trabalho busca minimizar o atraso total, que é um objetivo muito importante para sistemas de manufatura (Raman, 1995), já que datas de entrega são muito comuns nesses sistemas (Panwalkar, Smith e Seidmann, 1982) e atraso podem levar a insatisfação dos clientes e aumento dos custos (Sen e Gupta, 1984).

Além da configuração dos recursos e do critério de otimização, um problema de Programação também é definido por restrições adicionais específicas do problema. Neste trabalho é assumida uma restrição *FIFO (First In First Out)* para o problema, uma restrição que é amplamente usada para problemas de *Flow Shop* e que é condizente com práticas comuns da indústria de manufatura. A restrição *FIFO* implica que os *jobs* seguem a mesma ordem de processamento em todas as máquinas, e portanto apenas uma sequência permutacional dos *jobs* já representa toda a sequência de produção em todas as máquinas. Um *flow shop* com a restrição FIFO é chamado de *flow shop* permutacional.

Por fim, uma última restrição adotada no problema estudado neste trabalho é a existência de janelas de indisponibilidade. A maior parte da literatura de Programação da Produção parte da hipótese de que os recursos estão disponíveis durante todo o horizonte de tempo considerado, porém muitas vezes essa hipótese não é condiz com o cenário real de produção industrial. Segundo Ma, Chu e Zuo (2010) alguns cenários em que as

máquinas não estão disponíveis durante todo o período são por quebra de uma máquina, manutenção preventiva e também quando uma máquina está indisponível no início do período por ainda ter tarefas de um período anterior para finalizar. Neste trabalho, considera-se um *flow shop* com até um período de indisponibilidade em cada uma das máquinas.

Em problemas de Programação da Produção com janelas de indisponibilidade, pode-se considerar que uma tarefa pode começar a ser processada por uma máquina antes de seu período de indisponibilidade e ter seu processamento retomado pela máquina assim que a indisponibilidade termina, ou pode-se considerar que uma tarefa não pode ser interrompida por uma janela de indisponibilidade da máquina em que está sendo processada. Ambos os casos tem aplicação na indústria, dependendo do contexto específico de produção. Neste trabalho considera-se o segundo caso, que na literatura é conhecido como *non-resumable*.

1.2 Objetivo

O objetivo deste trabalho é estudar o problema de minimização do atraso total em um *flow shop* permutacional de duas máquinas com uma janela de indisponibilidade *non-resumable* em cada uma das máquinas. Com este enfoque será apresentada uma formulação de Programação Linear Inteira Mista desse problema, além disso três métodos heurísticos adaptados de métodos conhecidos da literatura serão propostos.

Como o problema estudado neste trabalho é NP-Difícil, é de grande importância estudar o resultado de métodos heurísticos, pois em muitas situações a resolução de instâncias grandes por métodos exatos é inviável em um tempo razoável para a aplicação. Sabe-se que o problema tratado neste trabalho é NP-Difícil dado que Du e Leung (1990) provaram que o problema de minimização do atraso em uma única máquina é NP-Difícil, e o problema do *flow shop* com janelas de indisponibilidade é uma generalização do problema em máquina única sem restrições de disponibilidade.

2 Revisão da Literatura e Descrição do Problema

Neste capítulo é feita uma revisão da literatura com uma visão geral de Programação da Produção, em seguida mais especificamente sobre problemas de *flow shop* com restrições de disponibilidade e problemas de *flow shop* com objetivo de minimizar o atraso.

2.1 Programação da Produção (*Scheduling*)

Pinedo (2012) define a Programação da Produção, ou *Scheduling*, como o exercício de alocar recursos para a realização de tarefas durante determinados períodos de tempo, otimizando um ou mais critérios. Os recursos a serem alocados e as tarefas a serem realizadas podem ser diversos, como máquinas em uma oficina para realizarem processos produtivos, pistas de um aeroporto a serem alocadas para pousos e decolagens ou ainda unidades de processamento de um computador para executarem programas de um computador. (Pinedo 2012). Esses recursos a serem alocados são normalmente chamados de 'máquinas' e as tarefas a serem realizadas de '*jobs*' (sendo que um *job* pode requerer uma ou mais operações a serem realizadas pelas máquinas).

Os problemas de Programação da Produção podem ser categorizados através de diversos critérios. O principal critério para uma categorização mais geral desses problemas é a presença de elementos aleatórios: problemas que não possuem elementos aleatórios são chamados determinísticos, enquanto os que possuem elementos aleatórios são chamados estocásticos.

Outro critério importante para categorizar os problemas de Programação da Produção é a configuração das máquinas. As principais configurações de máquinas são: máquina única, máquinas em paralelo, *flow shop*, *job shop*, e *open shop*, que são explicadas a seguir (Pinedo 2012):

Máquina única: O caso da máquina única é o mais simples dentre as configurações de máquinas, sendo um caso específico de todas as outras configurações de máquinas mais complexas. Um exemplo é apresentado na Figura 1.

Figura 1: Exemplo de *schedule* de máquina única com 3 *jobs*.

Máquinas em paralelo: Há *m* máquinas em paralelo. Cada *job* requer uma única operação a ser realizada por alguma das *m* máquinas (ou por um subconjunto dessas máquinas). Problemas nessa configuração de máquinas costumam ser categorizados ainda pela relação entre a velocidade de processamento de cada uma das máquinas paralelas, podendo ser máquinas idênticas, com velocidades idênticas (ou seja, seguem uma proporção no tempo de processamento que independe do *job* específico a ser processado), ou não relacionadas. Um exemplo é apresentado na Figura 2.

Figura 2: Exemplo de *schedule* de máquinas paralelas com 3 máquinas e 9 *jobs*.

***Flow shop*:** Há *m* máquinas em série. Cada *job* tem que ser processado em cada uma das *m* máquinas. Além disso, todos os *jobs* devem seguir o mesmo roteiro (ou seja, ser processado primeiro na máquina 1, depois na máquina 2, e assim por diante até a máquina *m*). Após concluir seu processamento em uma máquina, o *job* entra em uma fila para ser processado pela máquina seguinte. Problemas de *flow shop* normalmente possuem uma restrição '*Primeiro a Entrar, Primeiro a Sair*' (*PEPS*, ou *FIFO* de '*First In, First Out*'), que proíbe um *job* de passar na frente de outro na fila. Um *flow Shop* que possuí a restrição *FIFO* é chamado *flow shop* permutacional. Um exemplo é apresentado na Figura 3.

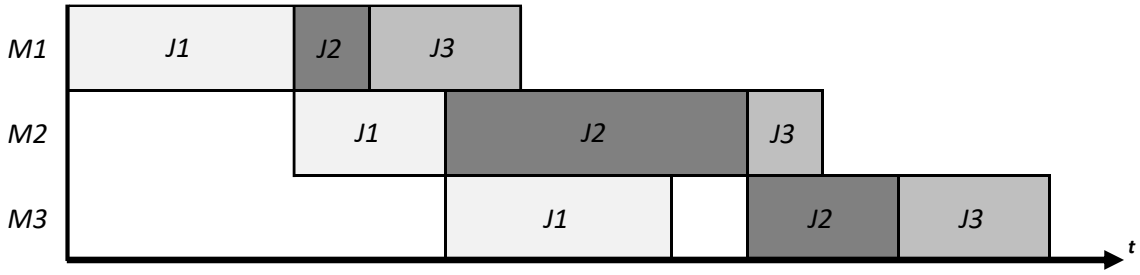

Figura 3: Exemplo de *schedule flow shop* com 3 máquinas e 3 jobs.

Job shop: Há *m* máquinas. Cada *job* tem o seu próprio roteiro determinado para seguir, devendo passar em ordem por uma determinada sequência de

máquinas. Dessa forma, o *job shop* é uma generalização do *flow shop.* Um exemplo é apresentado na Figura 4.

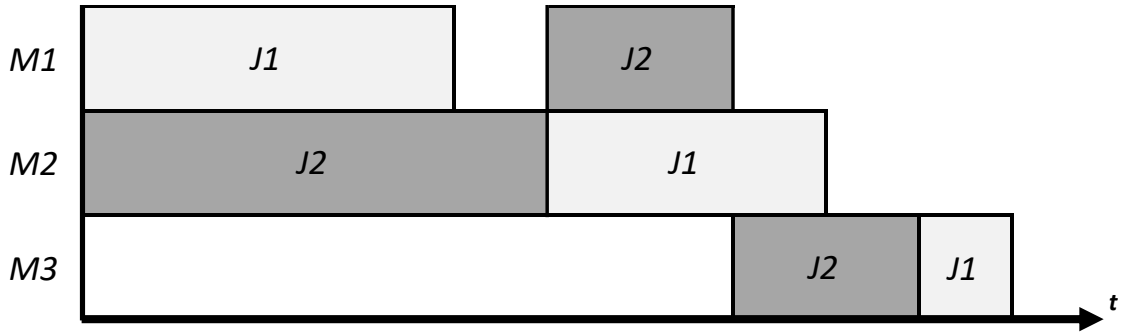

Figura 4*:* Exemplo de *job shop* com 3 máquinas e 2 *jobs*. Observa-se que os *jobs* tem roteiros diferentes: *J1* segue *M1 -> M2 -> M3* enquanto *J2* segue *M2 -> M1 -> M3*.

Open shop: Há *m* máquinas. Cada job deve ser processado por um sub-conjunto de máquinas, porém o roteiro que deve seguir não é predeterminado e é uma decisão a ser tomada durante a programação da produção. Um exemplo é apresentado na Figura 5.

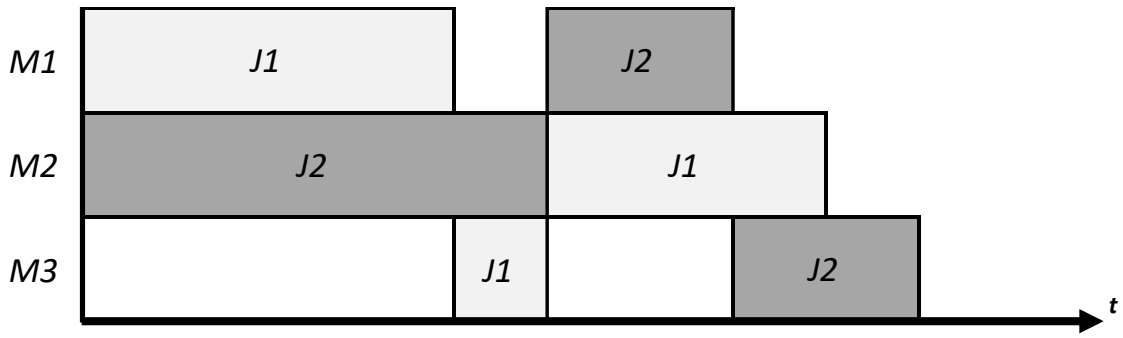

Figura 5: Exemplo de open shop com 3 máquinas e 2 *jobs*. Considerando os mesmos *jobs* e máquinas apresentados na figura 4 (*job shop*), aqui o planejador tem a liberdade de alterar o roteiro dos jobs. No caso, o roteiro do *J1* foi alterado para *M1 -> M3 -> M2*.

Dentre os trabalhos pioneiros de Programação da Produção, um destaque é o de Johnson (1954), que estuda o problema de minimização do *makespan* (instante de conclusão do último *job* processado) em *flow shops* de 2 ou 3 máquinas. Nesse trabalho ele apresenta uma regra simples que minimiza o *makespan* encontrando a solução ótima e que ficou conhecida como Algoritmo de Johnson.

Seguindo a notação do Pinedo (2012), um problema de programação da produção pode ser descrito por um trio de hiperparâmetros α|β|γ. α descreve a configuração das máquinas e usualmente tem só um elemento. β descreve

características e restrições adicionais do processamento e pode ter nenhum, um, ou vários elementos. γ descreve o objetivo a ser minimizado e usualmente só tem um elemento. Tomando como exemplo o problema $P3|prec|Cmax$, $P3$ indica que trata-se de um problema de máquinas paralelas com 3 máquinas, $prec$ indica que o problema inclui restrições de precedência (alguns *jobs* tem a restrição de ser feito depois de algum outro *job*) e $Cmax$ indica que o objetivo a ser minimizado é o *makespan* (maior tempo de conclusão de *job*).

2.2 Problemas com Restrição de Disponibilidade

Na literatura de *Scheduling* (Programação da Produção), é usual assumir a hipótese de que todas as máquinas estão disponíveis durante todo o horizonte de planejamento. Entretanto, em uma grande variedade de situações reais essa hipótese não é de fato verdadeira. Situações comuns em que não há completa disponibilidade das máquinas são horizontes com atividade de manutenção preventiva de uma ou mais máquinas e horizontes em que há possibilidade de quebra de alguma máquina. (Schmidt 2000).

Nos problemas com Restrição de Disponibilidade, há uma distinção entre diferentes tipos de restrição relacionados à possibilidade de um *job* retomar o seu processamento após ser interrompido por um período de indisponibilidade da máquina em que começou a ser processado. A definição de indisponibilidade *resumable* e *non-resumable* é feita por Lee (1996) e a de *semiresumable* por Lee(1999). O caso *resumable* significa que quando um processamento de um *job* é interrompido por indisponibilidade da máquina, ele pode ser retomado de onde parou (sem perda da fração já processada), enquanto o caso *non-resumable* significa que ele não pode ser retomado e deve ser reiniciado do zero. Já o caso *semiresumable* é uma generalização desses, definindo uma variável *alfa* entre 0 e 1 indicando a fração que deve ser reprocessada (do processamento interrompido por indisponibilidade de máquina), de forma que *alfa*=0 corresponde ao caso *resumable* e *alfa*=1 ao caso *non-resumable*. Um exemplo simples com apenas 1 *job* e comparando esses três casos é apresentado na Figura 6.

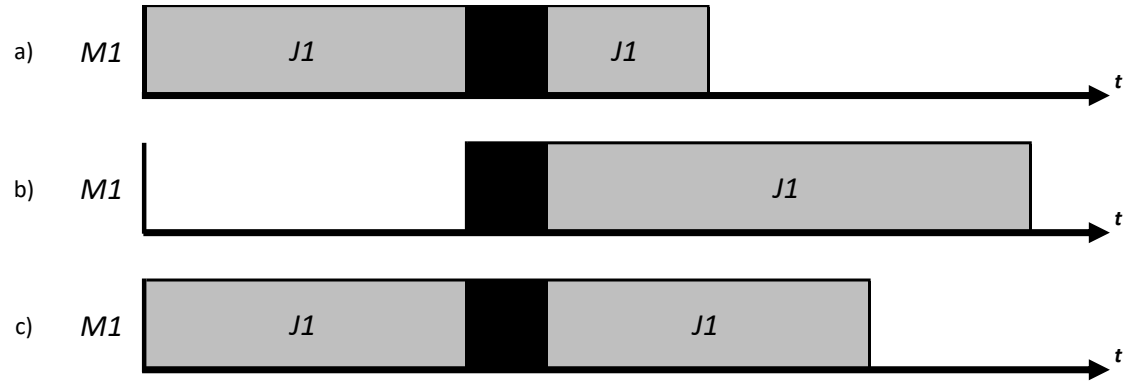

Figura 6: Representação dos diferentes tipos de indisponibilidade para um *schedule* de um *workshop* de uma máquina e um *job* (com tempo de processamento fixo) com diferentes tipos de restrição de disponibilidade: a) *resumable*, b) *non-resumable* e c) *semiresumable*. A cor preta representa período de indisponibilidade.

Adiri *et al*. (1989) estudam o problema de minimização de *flowtime* (soma dos tempos de finalização de todos os *jobs*) em uma máquina com um período de indisponibilidade (*non-resumable*), lidando tanto com o caso estocástico em que o momento de falha da máquina e o tempo de reparo são aleatórios como com o caso determinístico em que essas variáveis já são conhecidas antes do planejamento. Os autores mostram que o problema é NP-Difícil e estudam a aplicação da heurística *SPT* (*Shortest Processing Time* ou Menor Tempo de Processamento*)* nele. Esse problema foi também estudado por Lee e Liman (1992), que apresentam uma prova mais simples de que o problema é NP-Difícil e revisam a margem de erro relativa da aplicação do SPT. Sadfi et al. (2005) propõem um algoritmo *MSPT* (*SPT* Modificado) para esse problema, He, Zhong e Gu (2006) apresentam um algoritmo *PTAS* (*Polynomial Time Approximation Scheme)* e Breit (2007) apresenta um algoritmo paramétrico *O(n*log *n)* capaz de obter melhores margens de erro de pior caso.

Lee (1991) estuda o problema de minimização do *makespan* em um ambiente com *m* máquinas paralelas, em que cada máquina *i* só se torna disponível em um instante a_i arbitrário. Ele desenvolve uma heurística *MLPT* (*Modified Longest Processing Time* ou Maior Tempo de Processamento Modificado) para lidar com o problema. Lin, He, Yao e Lu (1997) estudam as margens de erro do algoritmo *MLPT* proposto por Lee (1991) e propõem mais duas variações de heurísticas *MLPT*. Esse problema é também estudado por Kellerer (1998), que desenvolve uma heurística de aproximação dual para o problema, e também estuda o mesmo problema com o tempo de conclusão mínimo C_{min} (instante em que alguma das máquinas acaba de processar todos os *jobs* a ela assignados) como função objetivo a ser maximizada. Chang e Hwang (1999) também estudam esse problema, aplicando a heurística *MULTIFIT* para resolvê-lo. Lu e Posner (1993) estudam o problema de minimização do *makespan* em um *Open Shop* de 2 máquinas com a restrição de que uma das máquinas só torna-se disponível em um instante *t* e propõem um método de resolução em tempo polinomial.

Lee e Liman (1993) estudam o problema de minimização do *flowtime* em um ambiente de duas máquinas paralelas, em que uma das máquinas só funciona por um determinado tempo e depois se torna indisponível. Eles mostram que o problema é NP-Difícil e apresentam um algoritmo de programação dinâmica pseudo-polinomial. Mosheiov (1994) estuda um problema similar com *m* máquinas, em que cada máquina tem um horizonte de disponibilidade $[x_j, y_j]$, e demonstra que a heurística *SPT* é assintoticamente ótima para o problema conforme o número de *jobs* aumenta.

Lee (1996) além de definir as indisponibilidades *resumable* e *non-resumable* estuda diversos problemas determinísticos com esses tipos de indisponibilidade com as configurações de máquina única e máquinas paralelas e com diferentes funções objetivo. Para cada problema ele apresenta um algoritmo ótimo de tempo polinomial ou uma prova de que o problema é NP-Difícil.

Hwang e Chang (1998) e Hwang, Lee e Chang (2005) estudam o problema de minimização de *makespan* em uma configuração de máquinas paralelas em que cada máquina pode ter um intervalo de indisponibilidade *non-resumable* e analisam os casos de pior performance do algoritmo *LPT (Longest Processing Time* ou Maior Tempo de Processamento). Eles concluem que o principal fator que afeta a performance no pior caso do algoritmo *LPT* é o número de máquinas que podem trabalhar simultaneamente e não o número de máquinas que podem ficar indisponíveis. Liao *et al.* (2005) estudam um caso específico deste problema, com duas máquinas em que apenas uma delas possui período de indisponibilidade, e apresenta um algoritmo baseado no algoritmo *TMO* (*Two Machine Optimization*) proposto por Ho e Wong (1995), que resolve o problema encontrando o ótimo. Lin e Liao (2007) generalizam o resultado de Liao *et al.* (2005) com a restrição de que ambas as máquinas podem ficar indisponíveis em vez de apenas uma, e apresentam um algoritmo baseado em busca lexicográfica que resolve o problema encontrando o ótimo.

Seguindo a notação α|β|γ apresentada no final da Seção 2.1, há alguns termos específicos usados para descrever problemas com restrição de disponibilidade: Ao hiperparâmetro α, que normalmente possui apenas um elemento, acrescenta-se outro elemento hjk descrevendo quantas janelas (índice *k*) há e em quais máquinas (índice *j*) (Ma, Chu, Zuo 2010). No hiperparâmetro β inclui-se um termo indicando o tipo de indisponibilidade entre *r – a*, *nr – a* e *sr – a,* que representam respectivamente janelas *resumable*, *non-resumable* e *semiresumable*. O termo *a* na expressão é abreviação de *availability constraint* (restrição de disponibilidade).

2.3 Flow Shop

Em vários ambientes de manufatura e montagem, cada *job* deve passar por uma série de operações. Frequentemente, essas operações dever ser feitas em todos os *jobs* na mesma ordem, implicando que todos os *jobs* devem seguir o mesmo roteiro. Então, assume-se que as máquinas são organizadas em série e o ambiente de produção é chamado de *flow shop*. (Pinedo 2012)

O artigo de Johnson (1954) apresentado na seção anterior, como um dos pioneiros na área de pesquisa de Programação da Produção também foi um dos pioneiros no estudo de *flow shop.*

2.3.1 *Flow Shop* com Restrição de Disponibilidade

Em relação a problemas de *flow shop* com restrições de disponibilidade, a survey Ma, Chu, Zuo (2010) afirmam que há muitos estudos para casos com duas máquinas, porém poucos para casos com múltiplas máquinas. Além disso, indicam que quase toda pesquisa na área tem como critério de otimização o *makespan*, e que o estudo do problema com diferentes critérios de otimização é uma direção interessante para futura pesquisa. Lee (1997) é pioneiro nessa área, estudando o problema de *flow shop* de duas máquinas com indisponibilidade (*resumable*) em uma das máquinas e buscando minimizar o *makespan*. Ele afirma que o problema é NP-Difícil para duas máquinas se alguma tiver um ‘buraco’ de disponibilidade, enquanto que o caso clássico sem restrição de disponibilidade pode ser resolvido em tempo polinomial pelo Algoritmo de Johnson proposto por Jonhson (1953).

Considerando o caso de minimização do *makespan* em um *flow shop* de duas máquinas com restrição de disponibilidade *resumable* apenas na primeira, Lee (1997) mostra que a aplicação do Algoritmo de Johnson tem margem de erro relativa menor ou igual a 1. Allaoui *et al.* (2006) estudaram a aplicação do Algoritmo de Johnson a esse problema e estabeleceram as condições nas quais esse algoritmo retorna a solução ótima. Lee (1997) também propôs uma heurística de tempo $O(n \log n)$ com margem de erro relativa de até 1/2. Cheng e Wang (2000) mostraram que essa margem de ½ era justa e propuseram uma heurística melhorada, com uma margem de erro de até 1/3 que mais tarde foi melhorada para 1/4 por Breit (2004). Ng e Kovalyov (2004) estudam propriedades de uma programação ótima para esse problema e propõem um esquema de aproximação em tempo totalmente polinomial (*FPTAS Fully Polynomial Time Approximation Scheme*). Mais tarde Wang e Cheng (2007a) e Wang e Cheng (2007b) estudam o problema considerando também o tempo de *setup* e desenvolvem um esquema de aproximação em tempo polinomial (*PTAS*

Polynomial Time Approximation Scheme) e uma heurística com margem de erro relativo menor ou igual a 2/3 para resolvê-lo.

Ainda, no caso de duas máquinas com restrição de disponibilidade *resumable*, mas com restrição de disponibilidade apenas na segunda máquina, Lee (1997) mostra que a aplicação do Algoritmo de Johnson tem margem de erro relativa menor ou igual a 1/2 e propõe uma heurística de tempo $O(n \log n)$ com margem de erro menor ou igual a 1/3. Similar ao caso de indisponibilidade na primeira máquina, Ng e Kovalyov (2004) também estudam as propriedades de uma programação ótima com indisponibilidade na segunda máquina. Wang e Cheng (2007b) estudam uma variante desse problema com tempo de setup e apresentam heurística com margem de erro relativo menor ou igual a 2/3 para resolvê-lo.

Breit (2006) estuda a variante preemptiva (permite interrupções arbitrárias nos processamentos) do problema com duas máquinas com um buraco de disponibilidade na primeira ou na segunda máquina, e propõe um *PTAS* para resolvê-lo.

Para o problema em que cada uma das duas máquinas pode ter um número arbitrário de buracos de disponibilidade, Błazewicz *et al.* (2001) apresentam duas heurísticas construtivas e uma heurística de busca local para resolvê-lo. Kubiak *et al.* (2002) consideram os problemas com vários buracos apenas na máquina 1, apenas na máquina 2 e em ambas, mostrando que o problema é NP-Difícil no forte sentido mesmo que apenas uma das máquinas tenha número arbitrário de buracos. Kubzin, Potts e Strusevich (2009) propõem uma heurística para o caso em que apenas M1 tem número arbitrário de buracos.

Lee (1999) estuda o *flow shop* de duas máquinas com indisponibilidade *semiresumable* abordando também os casos particulares *resumable* e *non-resumable*, apresenta um algoritmo de programação dinâmica pseudo-polinomial que resolve o problema encontrando o ótimo e fornece algumas heurísticas com análise margem de erro para elas.

Xu *et al.* (2018) apresentam diferentes formulações de programação linear inteira mista (*MILP - Mixed Integer Linear Programming*) para o *flow shop* de duas máquinas com uma restrição de disponibilidade em uma das máquinas e avaliam o desempenho de cada formulação na resolução de instâncias de problemas. Resultados numéricos mostraram que cada formulação podia resolver instâncias de até 100 *jobs* em tempo razoável.

Aggoune (2004) é pioneiro em estudar o *flow shop* de m máquinas com restrições de disponibilidade, considerando o problema com múltiplas indisponibilidades em cada máquina e minimizando o *makespan*. Ele mostra que o problema é NP-difícil no sentido forte, e por isso propõe uma heurística genética e uma de *tabu search* para resolver o problema. Mais tarde Aggoune e

Portmann (2006) apresentam uma abordagem geométrica temporizada (*temporizer geometric*) para resolver o problema com dois *jobs* e uma heurística para conseguir uma solução aproximada para o problema com mais *jobs*.

2.3.2 *Flow Shop* com Restrição de Disponibilidade e Função Objetivo de Atraso

O problema de minimização do atraso total em um *flow shop* é NP-Difícil. Chega-se facilmente a essa conclusão a partir do resultado de Du e Leung (1990) de que a minimização do atraso total em uma máquina é NP-Difícil, já que o problema com uma máquina é um caso específico de *flow shop*.

O critério de atraso total é importante para a indústria de manufatura, pois quando um *job* não é completado antes do seu prazo há alguns custos para a empresa (Armentano e Ronconi 1999). Segudo Sen e Gupta (1984), esses custos podem ser penalidades contratuais, dano à imagem da empresa que aumenta a chance de perder o cliente para futuros trabalhos e dano à reputação que diminui a chance de novos clientes buscarem a empresa.

Kim (1993) revisa a literatura inicial de *flow shops* minimizando o atraso total, apresentado diferentes abordagens heurísticas para resolver o problema. Armentano e Ronconi (1999) estudam a aplicação de uma heurística *tabu search* para resolver esse tipo de problema e obtêm resultados promissores em experimentos numéricos. Hasija e Rajendran (2004) propõem uma heurística baseada em recozimento simulado (*simulated annealing*) para o problema, com resultados bons em experimentos numéricos. Vallada, Ruiz e Minella (2008) fazem uma revisão da literatura de heurística para a minimização do atraso total no *flow shop* de m-máquinas. Framinan e Leisten (2008) desenvolvem como heurística um algoritmo guloso (greedy), com bons resultados nos experimentos numéricos. Vallada e Ruiz (2010) propõem algoritmos genéticos para lidar o problema, e esses algoritmos provaram-se eficazes nos resultados dos experimentos numéricos. Karabulut (2016) propõe um algoritmo guloso iterado para resolver o problema, obtendo resultados superiores a outros algoritmos gulosos.

Fernandez-Viagas, Valente e Framinan (2018) realizam uma abrangente avaliação das heurísticas e meta-heurísticas com melhores desempenhos para o problema e propõem uma heurística de *beam search* e um conjunto de algoritmos gulosos iterados com 8 diferentes procedimentos para a etapa de destruição e construção do algoritmo. Nos experimentos computacionais, a heurística *beam search* se destaca com desempenho superior às demais heurísticas construtivas, enquanto o algoritmo guloso iterado com o procedimento de troca adjacente aleatória teve o melhor desempenho entre as meta-heurísticas analisadas.

Ao conhecimento do autor, ainda não foram publicados artigos científicos lidando com o *flow shop* de 2 máquinas com restrição de indisponiblidade tendo como função objetivo o atraso total.

Neste trabalho são formuladas três variantes do problema de minimização do atraso em um *flow shop* de duas máquinas com restrições de disponibilidade *non-resumable*: a) Com uma janela de indisponibilidade na primeira máquina - $\mathrm{F2}, h_{11}|nr-a|\sum T_j$; b) Com uma janela de indisponibilidade na segunda máquina - $\mathrm{F2}, h_{21}|nr-a|\sum T_j$; c) Com uma janela de indisponibilidade em cada uma das máquinas - $\mathrm{F2}, h_{j1}|nr-a|\sum T_j$. Vale ressaltar que as variantes a) e b) são casos específicos da variante c).

3 Formulação Matemática

Nesse capítulo são formuladas utilizando programação linear inteira mista as três variantes do problema de minimização do atraso em um *flow shop* de duas máquinas com restrições de disponibilidade *non-resumable* apresentadas no final da seção 2.32: $\mathrm{F2}, h_{11}|nr-a|\sum T_j$, $\mathrm{F2}, h_{21}|nr-a|\sum T_j$ e $\mathrm{F2}, h_{j1}|nr-a|\sum T_j$.

3.1 Formulação do problema $\mathrm{F2}, h_{11}|nr-a|\sum T_j$

Nessa formulação baseada nas ideias de Wilson (1989) e de Xu *et al.*(2018), o período de indisponibilidade *non-resumable* é tratado como o *job* de índice *n*+1. Considere que s_1, $p_{1,n+1}$ e d_{n+1} são o instante de início da indisponibilidade na primeira máquina, a duração dessa indisponibilidade (que é o tempo de processamento do *job n*+1 na máquina 1) e a data de entrega para esse *job* de índice *n*+1. Esse job terá instante de início em s_1 e de término em $s_1 + p_{1,n+1}$. Note-se que $p_{2,n+1}$ (o tempo de processamento do *job n*+1 na máquina 2) vale 0, e que d_{n+1} será o término do período de indisponibilidade somado do maior tempo de processamento na segunda máquina, ou seja o instante $s_1 + p_{1,n+1} + max(p_{2j})$.

Variáveis:

x_{ij}: Variável binária que tem valor 1 se o *job i* foi programado como o *j*-ésimo *job*, e valor 0 caso contrário.

S_{rj}: Variável que indica o instante em que o *j*-ésimo *job* começa a ser processado pela máquina *r*

T_j: Variável que indica qual foi o atraso do *j*-ésimo *job*.

Parâmetros:

p_{ri}: Tempo de processamento do *job i* na máquina *r*.

d_i: Prazo de entrega para o *job i*.

s_1: Instante em que começa o período de indisponibilidade na máquina 1.

M: Valor suficientemente grande.

$$\min \sum_{j=1}^{n} T_j \tag{1}$$

$$\sum_{j=1}^{n+1} x_{ij} = 1 \qquad i = 1,2,\ldots,n+1 \tag{2}$$

$$\sum_{i=1}^{n+1} x_{ij} = 1 \qquad j = 1,2,\ldots,n+1 \tag{3}$$

$$S_{1j} - s_1 \leq M(1 - x_{n+1,j}) \qquad j = 1,2,\ldots,n+1 \tag{4}$$

$$S_{1j} - s_1 \geq -M(1 - x_{n+1,j}) \qquad j = 1,2,\ldots,n+1 \tag{5}$$

$$S_{r,j+1} \geq S_{r,j} + \sum_{i=1}^{n+1} p_{ri}\, x_{ij} \qquad j = 1,2,\ldots,n\,;\; r = 1,2 \tag{6}$$

$$S_{2,j} \geq S_{1,j} + \sum_{i=1}^{n+1} p_{1i}\, x_{ij} \qquad j = 1,2,\ldots,n+1 \tag{7}$$

$$S_{11} \geq 0 \tag{8}$$

$$T_j \geq S_{2j} + \sum_{i=1}^{n+1} x_{ij}\,(p_{2i} - d_i) \qquad j = 1,2,\ldots,n+1 \tag{9}$$

$$T_j \geq 0 \qquad j = 1,2,\ldots,n+1 \tag{10}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,2,\ldots,n+1 \tag{11}$$

A função objetivo (1) define como objetivo minimizar a soma do atraso de cada um dos *jobs* (considerando todas as posições dos *jobs* e do período de indisponibilidade). O conjunto de restrições (2) garante que cada *job* (incluindo a indisponibilidade) tem uma única posição na sequência de produção, enquanto o conjunto de restrições (3) garante que cada posição da sequência de produção está associada a um único *job* (ou à indisponibilidade). Os conjuntos de restrições (4) e (5) garantem que o período de indisponibilidade se iniciará em s_1. É interessante notar que nessas inequações, com o uso do M suficientemente grande, essas restrições apenas restringem de fato o espaço das soluções quando $x_{n+1,j}$ é igual 1, e nesse caso as restrições (4) e (5) ficam $S_{1j} - s_1 \leq 0$ e $S_{1j} - s_1 \geq 0$ respectivamente (com as quais chega-se a $S_{1j} = s_1$).

. A restrição (6) define que o processamento de um *job* em uma das máquinas só pode iniciar após o *job* anterior acabar de ser processado. A restrição (7) define que o processamento de um *job* na segunda máquina só pode começar após esse *job* acabar de ser processado pela primeira máquina. A restrição (8) define que o

primeiro *job* só pode começar a ser processado na primeira máquina a partir do instante 0. A restrição (9) define que o atraso do job na posição *j* é maior ou igual à diferença entre seu instante de conclusão e o seu prazo de entrega, enquanto a restrição (10) define que o atraso de cada posição não pode ser negativo. Por fim, a restrição (11) define a restrição binária para cada uma das variáveis x_{ij}.

3.2 Formulação do problema $\mathrm{F2}, h_{21}|nr-a|\sum T_j$

Também baseada nas ideia de Wilson (1989), e de Xu *et al.* (2018), a formulação com restrição de disponibilidade *non-resumable* na máquina 2 também trata o período de indisponibilidade como o *job* de índice *n*+1. Porém, esse período inicia em s_2 e terminando em $s_2 + p_{2,n+1}$ já que a restrição passou a ser na máquina 2. A formulação desse problema é muito similar à formulação com restrição apenas na máquina 1, com alterações nas restrições (4) e (5) referentes ao período de indisponibilidade. Note-se que $p_{1,n+1}$ vale 0, e que d_{n+1} será o término do período de indisponibilidade, ou seja o instante $s_2 + p_{2,n+1}$.

Variáveis:

x_{ij}: Variável binária que tem valor 1 se o *job i* foi programado como o *j*-ésimo *job*, e valor 0 caso contrário.

S_{rj}: Variável que indica o instante em que o *j*-ésimo *job* começa a ser processado pela máquina *r*

T_j: Variável que indica qual foi o atraso do *j*-ésimo *job*.

Parâmetros:

p_{ri}: Tempo de processamento do *job i* na máquina *r*.

d_i: Prazo de entrega para o *job i*.

s_2: Instante em que começa o período de indisponibilidade na máquina 2.

M: Valor suficientemente grande.

$$\min \sum_{j=1}^{n} T_j \tag{12}$$

$$\sum_{j=1}^{n+1} x_{ij} = 1 \qquad i = 1,2,\ldots,n+1 \tag{13}$$

$$\sum_{i=1}^{n+1} x_{ij} = 1 \qquad j = 1,2,\ldots,n+1 \tag{14}$$

$$S_{2j} - s_2 \leq M(1 - x_{n+1,j}) \qquad j = 1,2,\ldots,n+1 \tag{15}$$

$$S_{2j} - s_2 \geq -M(1 - x_{n+1,j}) \qquad j = 1,2,\ldots,n+1 \tag{16}$$

$$S_{r,j+1} \geq S_{r,j} + \sum_{i=1}^{n+1} p_{ri}\, x_{ij} \qquad j = 1,2,\ldots,n\,;\; r = 1,2 \tag{17}$$

$$S_{2,j} \geq S_{1,j} + \sum_{i=1}^{n+1} p_{1i}\, x_{ij} \qquad j = 1,2,\ldots,n+1 \tag{18}$$

$$S_{11} \geq 0 \tag{19}$$

$$T_j \geq S_{2j} + \sum_{i=1}^{n+1} x_{ij}\,(p_{2i} - d_i) \qquad j = 1,2,\ldots,n+1 \tag{20}$$

$$T_j \geq 0 \qquad j = 1,2,\ldots,n+1 \tag{21}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,2,\ldots,n+1 \tag{22}$$

A função objetivo (12) é a mesma que a (1). Os conjuntos de restrições (13), (14), (17), (18), (19), (20), (21) e (22) são os mesmas que os (2), (3), (6), (7), (8), (9), (10) e (11) respectivamente. Já os conjuntos de restrições (15) e (16) garantem que o período de indisponibilidade se iniciará em s_2. É interessante notar que nessas inequações, com o uso do M suficientemente grande, essas restrições apenas restringem de fato o espaço das soluções quando $x_{n+1,j}$ é igual 1.

3.3 Formulação do problema $\mathrm{F2}, h_{j1}|nr - a|\sum T_j$

Com base nessas duas formulações, é proposta uma formulação que generaliza as duas, havendo possibilidade de haver uma restrição de disponibilidade em qualquer uma das máquinas ou em ambas. Nessa formulação, o período de indisponibilidade na primeira máquina será tratado como o *job* de índice *n* + 1 e o e o período de indisponibilidade na máquina 2 será tratado como o *job* de índice *n* + 2.

Variáveis:

x_{ij}: Variável binária que tem valor 1 se o *job i* foi programado como o *j*-ésimo *job*, e valor 0 caso contrário.

S_{rj}: Variável que indica o instante em que o *j*-ésimo *job* começa a ser processado pela máquina *r*

T_j: Variável que indica qual foi o atraso do *j*-ésimo *job*.

Parâmetros:

p_{ri}: Tempo de processamento do *job i* na máquina *r*.

d_i: Prazo de entrega para o *job i*.

s_1: Instante em que começa o período de indisponibilidade na máquina 1.

s_2: Instante em que começa o período de indisponibilidade na máquina 2.

M: Valor suficientemente grande.

$$\min \sum_{j=1}^{n} T_j \tag{23}$$

$$\sum_{j=1}^{n+1} x_{ij} = 1 \qquad i = 1,2,\ldots,n+2 \tag{24}$$

$$\sum_{i=1}^{n+1} x_{ij} = 1 \qquad j = 1,2,\ldots,n+2 \tag{25}$$

$$S_{1j} - s_1 \leq M(1 - x_{n+1,j}) \qquad j = 1,2,\ldots,n+2 \tag{26}$$

$$S_{1j} - s_1 \geq -M(1 - x_{n+1,j}) \qquad j = 1,2,\ldots,n+2 \tag{27}$$

$$S_{2j} - s_2 \leq M(1 - x_{n+2,j}) \qquad j = 1,2,\ldots,n+2 \tag{28}$$

$$S_{2j} - s_2 \geq -M(1 - x_{n+2,j}) \qquad j = 1,2,\ldots,n+2 \tag{29}$$

$$S_{r,j+1} \geq S_{r,j} + \sum_{i=1}^{n+1} p_{ri}\, x_{ij} \qquad j = 1,2,\ldots,n+1\,; \; r = 1,2 \tag{30}$$

$$S_{2,j} \geq S_{1,j} + \sum_{i=1}^{n+1} p_{1i}\, x_{ij} \qquad j = 1,2,\ldots,n+2 \tag{31}$$

$$S_{11} \geq 0 \tag{32}$$

$$T_j \geq S_{2j} + \sum_{i=1}^{n+1} x_{ij}\,(p_{2i} - d_i) \qquad j = 1,2,\ldots,n+2 \tag{33}$$

$$T_j \geq 0 \qquad j = 1,2,\ldots,n+2 \tag{34}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,2,\ldots,n+2 \tag{35}$$

A função objetivo (23) é a mesma que a (1). Os conjuntos de restrições (24), (25), (26), (27), (30), (31), (32), (33), (34) e (35) são os mesmos que os (2), (3), (4), (5) (6), (7), (8), (9), (10) e (11) respectivamente do problema com restrição apenas na primeira máquina. Já os conjuntos de restrições (28) e (29) são os mesmos que as (15) e (16) do problema com restrição na segunda máquina.

Nesses três problemas apresentados, é usado um parâmetro M 'suficientemente grande', ou seja, que sempre tem um valor superior ao módulo do lado esquerdo da equação em que se encontra. Para todos esses problemas, um valor adequado de M é a soma do instante de início do período de indisponibilidade que começa mais tarde com o maior tempo de processamento de algum job em qualquer das máquinas: $max(s_1, s_2) + max(p_{ri})\ (i = 1,2,\ldots,n+2\ , r = 1,2)$.

3.4 Instâncias exemplo

Para ilustrar o funcionamento dos modelos propostos foram criadas algumas instâncias exemplo, com 6 *jobs* cada, para apresentar uma solução ótima em cada modelo. Note-se que um parâmetro t_r é utilizado para referir-se ao término do período de indisponibilidade na máquina *r*.

Os dados das instâncias exemplo são apresentados na Tabela 1.

Tabela 1: Tabela dos *jobs* da instância exemplo, com seus tempos de processamento e data de entrega.

$\#J_i$	p_{1i}	p_{2i}	d_i
J_1	5	2	11
J_2	3	3	15
J_3	4	2	16
J_4	6	1	20
J_5	2	4	22
J_6	3	3	26

Além dos *jobs* na Tabela 1, a instância exemplo 1 tem uma janela de indisponibilidade na primeira máquina (*M1*) entre $s_1 = 9$ e $t_1 = 10$.

Resolvendo essa instância através do modelo de programação linear inteira mista (1) com o *solver Gurobi*, chegou-se à sequência ótima: (J1, J3, Indisponibilidade M1, J2, J5, J4, J6), com atraso total de 4.

Figura 7: Gráfico de Gantt da solução da instância exemplo 1. A cor preta representa período de indisponibilidade.

A instância exemplo 2 possui os mesmos *jobs*, porém em vez de ter indisponibilidade na *M1* possui indisponibilidade na *M2*, tendo como parâmetros $s_2 = 15$ e $t_2 = 16$. Resolvendo o problema através do modelo (2) chega-se à solução ótima: (J2, J1, J3, Indisponibilidade, J5, J4, J6), com atraso total de 1.

Figura 8: Gráfico de Gantt da solução da instância exemplo 2. A cor preta representa período de indisponibilidade.

Já a instância exemplo 3 possui os mesmos *jobs* das instâncias anteriores, e possui tanto a indisponibilidade na *M1* como na *M2* (com os mesmos parâmetros). Resolvendo o problema através do modelo (3) chega-se à solução ótima: (J2, J1, Indisponibilidade M1, Indisponibilidade M2, J3, J5, J4, J6), com atraso total de 7.

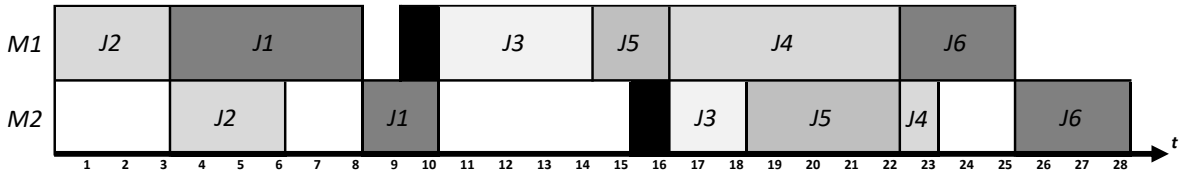

Figura 9: Gráfico de Gantt da solução da instância exemplo 3. A cor preta representa período de indisponibilidade.

4 Proposta de Heurísticas Eficientes

Como o problema de minimização do atraso total no *flow shop* de duas máquinas com restrições de disponibilidade é NP-Difícil, pode ser inviável resolver o problema com métodos exatos em um tempo aceitável. Por isso, neste capítulo proponho heurísticas adaptadas da literatura para resolver o problema em um tempo reduzido.

4.1 Heurística EDD (*Earliest Due Date)*

A heurística EDD, uma das principais utilizadas para problemas de *Scheduling* de minimização do atraso, consiste em ordenar os *jobs* em ordem não decrescente pela data de entrega.

4.2 Heurística NEH

A heurística NEH, introduzida no artigo Nawaz *et al*. (1983), é uma das principais heurísticas construtivas utilizadas para problemas de *flow shop* permutacional, em especial para problemas de minimização do *makespan*. Essa heurística é constituída por 2 etapas:

Etapa 1: Todos os *jobs* são ordenados segundo algum critério, por exemplo o LPT (Maior tempo de processamento).

Etapa 2: Seguindo a ordem da etapa 1, os *jobs* são adicionados um a um à solução, inseridos na posição que maximiza (ou minimiza) uma dada função objetivo para a sequência da solução.

Para resolver os problemas propostos nesse trabalho, será adotada a heurística NEH com os critérios:

Para a etapa 1, os *jobs* são ordenados em ordem não decrescente de um limitante inferior $LI_i = d_i - p_{1i} - p_{2i}$. Esse limitante inferior representa o valor mínimo de atraso de cada tarefa e foi introduzido na literatura como um componente da regra de despacho *MDD (Modified Due Date)* de Baker e Bertrand (1982). Esse limitante foi usado (no caso geral para *m* máquinas) também para ordenar os *jobs* por Amentano e Ronconi (1999).

Para a etapa 2, o critério para selecionar a posição em que o *job* será inserido foram utilizados dois critérios: Menor atraso total (da sequência parcial) e um critério híbrido do atraso total com o *makespan*. O critério atraso total é a soma dos atrasos de todos os *jobs* incluídos na sequência parcial, o que coincide com a função objetivo quando todos os *jobs* foram incluídos na sequência. O critério híbrido é o mesmo que o critério de atraso total, porém em caso de empate no atraso total o algoritmo seleciona a posição que minimiza o *makespan.* Nos próximos capítulos, a

heurística NEH com critério de atraso total e a heurística NEH com critério híbrido serão referidas como NEH-T e NEH-H respectivamente.

Para ilustrar o funcionamento da heurística NEH-H, considere a instância exemplo 3 – com os *jobs* da Tabela 1 e janelas de indisponibilidade entre $s_1 = 9$ e $t_1 = 10$ na primeira máquina (*M1*) e entre $s_2 = 15$ e $t_2 = 16$ na segunda máquina (*M2*). Realizando a etapa 1 nessa instância, calcula-se o LI_i de cada *job* e em seguida ordenam-se os *jobs* em ordem não decrescente de LI_i. O resultado dessa etapa 1 é apresentado na Tabela 2. Ao final da etapa 1 a ordenação dos *jobs* é $(J_1, J_2, J_3, J_4, J_5, J_6)$ e será utilizada na etapa 2.

Tabela 2: Jobs da instância exemplo ordenados em ordem não decrescente pelo Limite Inferior do Atraso

$\#J_i$	p_{1i}	p_{2i}	d_i	LI_i
J_1	5	2	11	**4**
J_2	3	3	15	**9**
J_3	4	2	16	**10**
J_4	6	1	20	**13**
J_5	2	4	22	**16**
J_6	3	3	26	**20**

Na etapa 2, cada *job* será inserido iterativamente na solução. Seguindo a ordem da Tabela 2, primeiro adiciona-se o J_1. Em seguida adiciona-se o J_2, e para isso avalia-se qual o atraso total T e qual o *makespan* C_{max} para cada uma das posições possíveis para inserção. Se o J_2 for inserido na primeira posição (formando a sequência J_2, J_1) o T será 0 e o C_{max} será 10, e se for inserido na segunda posição (formando a sequência J_1, J_2) o T será 0 e o C_{max} será 11. Pelo critério de inserção híbrido usado no NEH-H, o J_2 será inserido na primeira posição, pois o atraso T empatou e o C_{max} é menor com a inserção na primeira posição.

Em seguida insere-se o *job* J_3 e avaliam-se as sequências (J_3, J_2, J_1), (J_2, J_3, J_1) e (J_2, J_1, J_3) , cada uma relativa a uma possível inserção do J_3. Essas possíveis inserções são ilustradas na Figura 10. A sequência com o menor T é (J_2, J_1, J_3) que possui atraso total igual a 2, então o J_3 é inserido na terceira posição.

Esse processo é repetido até o último *job*, que nesse exemplo seria o J_6, ser inserido à solução. Seguindo a heurística NEH-H chega-se à sequência $(J_2, J_1, J_3, J_5, J_4, J_6)$ com atraso total igual a 7. Nesse caso, a sequência gerada pela heurística NEH-H é igual à solução ótima do problema apresentada na Figura 9,

porém não há garantia de que a sequência gerada pela heurística será a solução ótima.

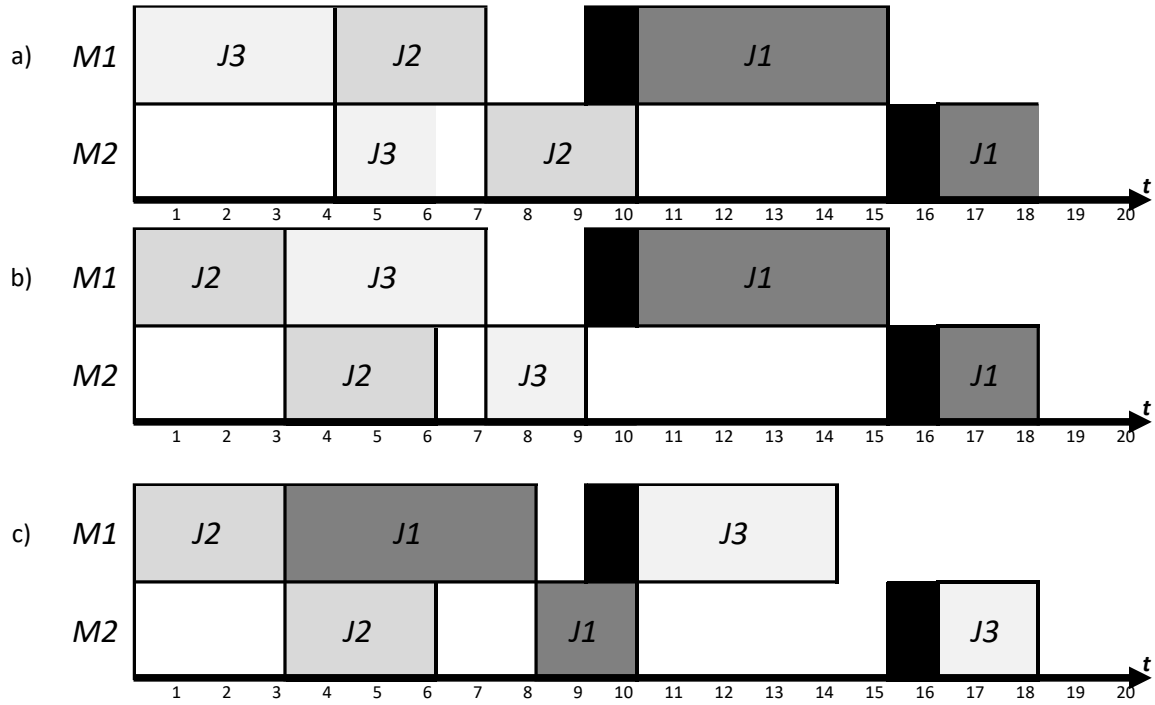

Figura 10: Representação em gráfico de Gantt de cada uma das três possíveis inserções do *job* J_3: a) na primeira posição, b) na segunda posição e c) na terceira posição.

5 Experimentos Computacionais

Foi realizada uma análise geral comparando resultados e tempo de execução do método exato, das heurísticas NEH (com critério de inserção de atraso total e com critério de inserção híbrido) e da heurística EDD.

Todos os experimentos foram executados em um computador com processador Intel Core i5 de 8GB de memória e implementados com linguagem Python 3.8. As formulações propostas foram resolvidas utilizando o *solver Gurobi* versão 9.1.0 com as configurações padrão.

Nas instâncias executadas pelo *Gurobi*, definiu-se o tempo máximo de execução em 30 minutos. Quando o programa não conseguir chegar à solução ótima nesse tempo, utilizou-se para comparação a melhor solução encontrada e foi incluído o limite inferior (*lower bound)* encontrado.

5.1 Instâncias

Para avaliar o impacto das janelas de indisponibilidade no atraso total e a performance das heurísticas propostas, foram realizados experimentos em instâncias de diversos tamanhos, e com diferentes situações de indisponibilidade de máquinas.

Para uma análise geral comparando os resultados dos métodos exatos com os das heurísticas propostas, foram geradas instâncias com número de *jobs* 5, 10, 15 20, 30, 50, 60, 70, 80, 90, 100 (seguindo Xu *et al.* 2018). Para cada um desses tamanhos do problema, foram geradas 5 instâncias com os tempos de processamento em cada uma vindos de uma distribuição discreta uniforme entre 1 e 100. Foram definidos períodos de indisponibilidade entre $s_r = (\sum_{i=1}^{n} p_{ri})/2$ e $t_r = s_r + 10$ (também seguindo Xu *et al.* 2018).

Seguindo Armentano e Ronconi (1999), para as datas de entrega foi utilizada uma distribuição uniforme de inteiros entre $P(1 - T - R/2)$ e $P(1 - T + R/2)$, em que P é um limite inferior do *makespan*, T é o fator de atraso e R é a amplitude de dispersão das datas de entrega. O limite inferior do *makespan* usado é $P = \min(p_{1i}) + (\sum_{i=1}^{n} p_{2i}) + t_2 - s_2$, que é uma adaptação de Pan *et al.* (2002) com o acréscimo do tempo de indisponibilidade da segunda máquina. Para a análise geral foram ultilizados como parâmetros $T = 0.4$ e $R = 0.6$, sendo esses valores os utilizados para o cenário de alto fator de atraso e baixa amplitude das datas de entrega em Armentano e Ronconi (1999) .

Para uma análise de sensibilidade, foram geradas instâncias com 15 *jobs* e variando os valores dos parâmetros T e R. Foram usados 0.3, 0.4 e 0.5 como valores de T e 0.5, 0.6 e 0.7 como valores de R. Para cada combinação de T e R

foram geradas 50 instâncias, em que os tempos de processamento e os períodos de indisponibilidade seguiram o modelo utilizado para a análise geral.

5.2 Análise Geral

Na Tabela 3 são apresentados os resultados da análise geral com o método exato. Nessa tabela, a coluna *Upper Bound* indica o atraso total na melhor solução encontrada, sendo essa solução ótima nas instâncias em que esse valor coincide com o *Lower Bound*. Para cálculo do *gap* foi utilizada a fórmula: $Gap(\%) = \frac{Upper\ Bound - Lower\ Bound}{Upper\ Bound}$

Na Tabela 4 são apresentados os resultados da análise geral com os métodos heurísticos, e na Tabela 5 são apresentados os tempos de execução com cada um dos métodos.

Pode-se observar que o método exato foi capaz de encontrar a solução ótima em menos de 3 segundos para todas as instâncias com até 15 *jobs*, e em menos de 30 minutos para todas as instâncias com até 30 *jobs*. Já as heurísticas NEH-T e NEH-H e EDD tiveram tempo de execução inferior 2 segundos para todas instâncias, sendo que o NEH-T e NEH-H tiveram tempo muito próximo em todas as instâncias e o EDD teve tempo inferior a 0,01 segundo em todas as instâncias.

Comparando os resultados das heurísticas com o *Upper Bound* encontrado pelo método exato, observa-se que o *gap* médio das heurísticas NEH-T, NEH-H e EDD foram 15,2%, 9,4% e 49,3% respectivamente. Ademais, das 55 instâncias analisadas, essas heurísticas chegaram a um *gap* inferior a 5% em 15, 22 e 1 instâncias respectivamente. Também é interessante notar que apenas a heurística NEH-H foi capaz de superar o *Gurobi* em algumas instâncias (em que o *Gurobi* não foi capaz de chegar na solução ótima em até 30 minutos), 4 instâncias ao todo com *Gap* médio de -1,4%.

Considerando esses resultados, conclui-se que a heurística NEH-H teve o melhor desempenho e é executável em tempo muito baixo (inferior a 2 segundos) para instâncias de até 100 *jobs*.

Tabela 3: Resultados da análise geral com o método exato

# Jobs	Instância	Upper Bound	Lower Bound	Gap (%)	Tempo de Execução (s)
5	1	395	395	0,00	0,08
5	2	676	676	0,00	0,05
5	3	184	184	0,00	0,06
5	4	429	429	0,00	0,11

5	5	140	140	0,00	0,06
10	1	618	618	0,00	0,29
10	2	747	747	0,00	0,31
10	3	96	96	0,00	0,27
10	4	876	876	0,00	0,23
10	5	571	571	0,00	0,27
15	1	736	736	0,00	1,51
15	2	1212	1212	0,00	1,52
15	3	1440	1440	0,00	0,39
15	4	756	756	0,00	0,56
15	5	408	408	0,00	2,58
20	1	1718	1718	0,00	17,69
20	2	343	343	0,00	3,95
20	3	1851	1851	0,00	581,19
20	4	2125	2125	0,00	0,78
20	5	1973	1973	0,00	4,68
30	1	2643	2643	0,00	5,12
30	2	1928	1928	0,00	12,83
30	3	2383	2383	0,00	59,41
30	4	1722	1722	0,00	419,25
30	5	1385	1385	0,00	1695,79
50	1	1430	1429	0,07	1800,00
50	2	5894	5256	10,82	1800,00
50	3	6550	5859	10,55	1800,00
50	4	995	995	0,00	361,33
50	5	5515	5039	8,63	1800,00
60	1	1897	1401	26,15	1800,00
60	2	7088	6199	12,54	1800,00
60	3	1642	1377	16,14	1800,00
60	4	1133	1113	1,77	19,14
60	5	2580	2580	0,00	1557,7
70	1	4265	3168	25,72	1800,00
70	2	2492	1914	23,19	1800,00
70	3	3994	3591	10,09	1800,00
70	4	3176	3059	3,68	1800,00
70	5	9792	9166	6,39	1800,00
80	1	13145	12074	8,15	1800,00
80	2	2636	2531	3,98	1800,00
80	3	12398	10697	13,72	1800,00
80	4	2400	2400	0,00	159,87
80	5	2333	2333	0,00	1233,49
90	1	9965	9166	8,02	1800,00
90	2	3136	3136	0,00	90,36
90	3	6330	4713	25,55	1800,00
90	4	11221	10887	2,98	1800,00

90	5	7540	6659	11,68	1800,00
100	1	5073	4768	6,01	1800,00
100	2	5825	5236	10,11	1800,00
100	3	8153	5652	30,68	1800,00
100	4	26988	24796	8,12	1800,00
100	5	7171	4786	33,26	1800,00
Média				5,78	
Desvio Padrão				8,93	

Tabela 4: Resultados da análise geral com os métodos heurísticos em comparação com o método exato.

# Jobs	Instância	Upper Bound	NEH-T	NEH-H	EDD	Gap NEH-T (%)	Gap NEH-H (%)	Gap EDD (%)
5	1	395	395	395	495	0,0	0,0	20,2
5	2	676	676	676	907	0,0	0,0	25,5
5	3	184	187	187	285	1,6	1,6	35,4
5	4	429	429	429	757	0,0	0,0	43,3
5	5	140	140	140	141	0,0	0,0	0,7
10	1	618	628	628	741	1,6	1,6	16,6
10	2	747	852	852	1093	12,3	12,3	31,7
10	3	96	96	96	570	0,0	0,0	83,2
10	4	876	917	931	1545	4,5	5,9	43,3
10	5	571	577	577	1346	1,0	1,0	57,6
15	1	736	858	854	834	14,2	13,8	11,8
15	2	1212	1512	1250	1786	19,8	3,0	32,1
15	3	1440	1755	1818	2207	17,9	20,8	34,8
15	4	756	821	784	1781	7,9	3,6	57,6
15	5	408	1012	866	1677	59,7	52,9	75,7
20	1	1718	1777	1777	2495	3,3	3,3	31,1
20	2	343	470	390	526	27,0	12,1	34,8
20	3	1851	1999	1999	4700	7,4	7,4	60,6
20	4	2125	2376	2373	2550	10,6	10,5	16,7
20	5	1973	2410	2385	3796	18,1	17,3	48,0
30	1	2643	2695	2774	3833	1,9	4,7	31,0
30	2	1928	2344	2344	3061	17,7	17,7	37,0
30	3	2383	2945	2933	6327	19,1	18,8	62,3
30	4	1722	2279	1991	3825	24,4	13,5	55,0
30	5	1385	1499	1471	2497	7,6	5,8	44,5
50	1	1430	2557	1733	4331	44,1	17,5	67,0
50	2	5894	7387	6998	12089	20,2	15,8	51,2

50	3	6550	8396	6837	11586	22,0	4,2	43,5
50	4	995	1371	1045	2872	27,4	4,8	65,4
50	5	5515	5842	5771	9966	5,6	4,4	44,7
60	1	1897	4782	2418	8651	60,3	21,5	78,1
60	2	7088	7091	6932	13569	0,0	-2,3	47,8
60	3	1642	2252	2058	4846	27,1	20,2	66,1
60	4	1133	1182	1229	3763	4,1	7,8	69,9
60	5	2580	3235	2747	6385	20,2	6,1	59,6
70	1	4265	4677	4389	14144	8,8	2,8	69,8
70	2	2492	2868	2470	12096	13,1	-0,9	79,4
70	3	3994	5387	5854	14878	25,9	31,8	73,2
70	4	3176	3478	3383	7202	8,7	6,1	55,9
70	5	9792	10493	10396	21203	6,7	5,8	53,8
80	1	13145	13971	14324	22149	5,9	8,2	40,7
80	2	2636	3663	3102	5676	28,0	15,0	53,6
80	3	12398	12531	12351	21970	1,1	-0,4	43,6
80	4	2400	3228	2647	4525	25,7	9,3	47,0
80	5	2333	4412	2534	5640	47,1	7,9	58,6
90	1	9965	11826	11690	22286	15,7	14,8	55,3
90	2	3136	3425	3176	6592	8,4	1,3	52,4
90	3	6330	7063	7295	19047	10,4	13,2	66,8
90	4	11221	11718	11659	18538	4,2	3,8	39,5
90	5	7540	9861	9504	15420	23,5	20,7	51,1
100	1	5073	9220	5932	15316	45,0	14,5	66,9
100	2	5825	6862	6725	12583	15,1	13,4	53,7
100	3	8153	9095	8012	19434	10,4	-1,8	58,0
100	4	26988	28653	28651	45047	5,8	5,8	40,1
100	5	7171	8883	8763	21427	19,3	18,2	66,5
Média						15,2	9,4	49,3
Desvio Padrão						14,6	9,6	18,0

Tabela 5: Comparação dos tempos de execução do método exato e dos métodos heurísticos.

# *Jobs*	**Instância**	**Tempo de Execução *Gurobi* (s)**	**Tempo de Execução NEH-T (s)**	**Tempo de Execução NEH-H (s)**	**Tempo de Execução EDD (s)**
5	1	0,08	< 0,01	< 0,01	< 0,01
5	2	0,05	< 0,01	< 0,01	< 0,01
5	3	0,06	< 0,01	< 0,01	< 0,01
5	4	0,11	< 0,01	< 0,01	< 0,01
5	5	0,06	0,02	< 0,01	< 0,01

10	1	0,29	< 0,01	< 0,01	< 0,01
10	2	0,31	< 0,01	< 0,01	< 0,01
10	3	0,27	< 0,01	< 0,01	< 0,01
10	4	0,23	0,02	< 0,01	< 0,01
10	5	0,27	< 0,01	< 0,01	< 0,01
15	1	1,51	< 0,01	< 0,01	< 0,01
15	2	1,52	< 0,01	0,02	< 0,01
15	3	0,39	< 0,01	< 0,01	< 0,01
15	4	0,56	< 0,01	0,02	< 0,01
15	5	2,58	< 0,01	< 0,01	< 0,01
20	1	17,69	< 0,01	0,02	< 0,01
20	2	3,95	< 0,01	0,02	< 0,01
20	3	581,19	0,02	0,02	< 0,01
20	4	0,78	0,02	0,02	< 0,01
20	5	4,68	0,02	0,02	< 0,01
30	1	5,12	0,03	0,03	< 0,01
30	2	12,83	0,03	0,03	< 0,01
30	3	59,41	0,03	0,03	< 0,01
30	4	419,25	0,05	0,03	< 0,01
30	5	1695,79	0,03	0,03	< 0,01
50	1	1800,00	0,16	0,14	< 0,01
50	2	1800,00	0,14	0,14	< 0,01
50	3	1800,00	0,14	0,14	< 0,01
50	4	361,33	0,16	0,14	< 0,01
50	5	1800,00	0,16	0,16	< 0,01
60	1	1800,00	0,23	0,25	< 0,01
60	2	1800,00	0,25	0,25	< 0,01
60	3	1800,00	0,25	0,25	< 0,01
60	4	19,14	0,25	0,25	< 0,01
60	5	1557,70	0,25	0,25	< 0,01
70	1	1800,00	0,39	0,39	< 0,01
70	2	1800,00	0,39	0,38	< 0,01
70	3	1800,00	0,39	0,38	< 0,01
70	4	1800,00	0,38	0,39	< 0,01
70	5	1800,00	0,39	0,39	< 0,01
80	1	1800,00	0,59	0,56	< 0,01
80	2	1800,00	0,58	0,58	< 0,01
80	3	1800,00	0,58	0,58	< 0,01
80	4	159,87	0,59	0,59	< 0,01
80	5	1233,49	0,61	0,58	< 0,01
90	1	1800,00	0,88	0,81	< 0,01
90	2	90,36	0,84	0,81	< 0,01
90	3	1800,00	0,81	0,80	< 0,01
90	4	1800,00	0,81	0,83	< 0,01
90	5	1800,00	0,83	0,83	< 0,01
100	1	1800,00	1,13	1,14	< 0,01
100	2	1800,00	1,14	1,13	< 0,01

100	3	1800,00	1,14	1,14	< 0,01
100	4	1800,00	1,14	1,13	< 0,01
100	5	1800,00	1,13	1,14	< 0,01

5.3 Análise de Sensibilidade

Na Figura 11 são representados os 9 cenários de parâmetros utilizados para a geração das datas de entrega, usando 0,3, 0,4 e 0,5 como valores de T e 0,5, 0,6 e 0,7 como valores de R. Com um parâmetro T maior, as datas de entrega ficam menores (os *jobs* devem ser entregues mais cedo para não incorrerem em atraso) e como consequência o atraso total deve aumentar. Já com um parâmetro R maior, as datas de entrega têm maior amplitude e variância.

Foi realizada uma análise de sensibilidade para o problema variando os parâmetros T e R e avaliando as diferenças no *gap* entre a solução ótima e a solução encontrada pela heurística NEH-H, a heurística que apresentou a melhor performance na análise geral. Na Tabela 6 são apresentados os resultados do *gap* médio em unidades de tempo e na Tabela 7 são apresentados os resultados do *gap* percentual médio.

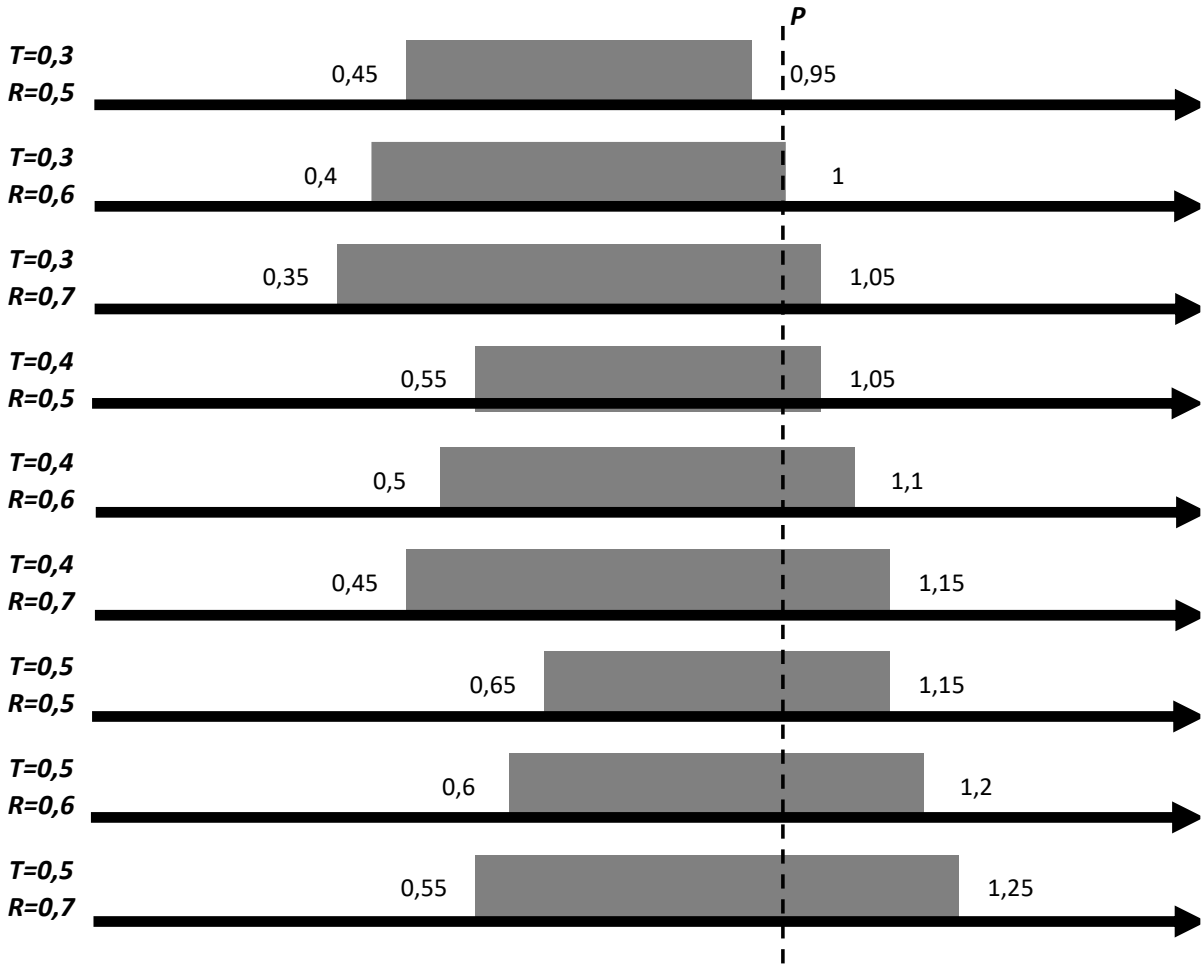

Figura 11: Representação dos cenários considerados para a geração das datas de entrega. *P* é o limitante inferior do *makespan*.

Pode-se observar um aumento significativo no *gap* em unidades de tempo conforme o parâmetro T aumenta, porém o *gap* percentual diminui. Já quando o parâmetro R aumenta, pode-se observar um aumento sutil no *gap* médio tanto em unidades de tempo como em porcentagem, porém esse aumento é pouco significativo.

Tabela 6: Resultados da análise de sensibilidade, considerando o *Gap* (em unidades de tempo) médio entre a solução ótima e a solução da heurística NEH-H.

T / R	**0,5**	**0,6**	**0,7**
0,3	89	116	89
0,4	116	160	153
0,5	228	212	259

Tabela 7: Resultados da análise de sensibilidade, considerando o *Gap* (em porcentagem) médio entre a solução ótima e a solução da heurística NEH-H.

T / R	0,5	0,6	0,7
0,3	19.3%	24.0%	21.7%
0,4	14.2%	17.1%	18.0%
0,5	14.3%	13.2%	14.8%

6 Conclusão

Este trabalho propôs estudo do problema de minimização do atraso total em um *flow shop* permutacional de duas máquinas com uma janela de indisponibilidade *non-resumable* em cada uma das máquinas. O problema foi formulado como um modelo de Programação Linear Inteira Mista e implementado com o *solver Gurobi* na linguagem Python buscar a solução ótima do problema.

Como o problema estudado neste trabalho é NP-Difícil, não foi possível gerar a solução ótima para a maioria das instâncias com 50 *jobs* ou mais em tempo de até 30 minutos, e tornou-se importante propor métodos heurísticos para o problema. Foram propostas duas heurísticas construtivas baseadas no algoritmo NEH (NEH-T com critério de atraso e NEH-H com critério híbrido do atraso e do *makespan*) e a aplicação da regra de despacho EDD (*Earliest Due Date*). Dessas heurísticas propostas, a que teve melhor desempenho foi a NEH-H com *gap* médio de 9,4% nas 55 instâncias analisadas.

Como direção para pesquisas futuras desse problema, pode ser relevante a proposição de novos métodos heurísticos e meta-heurísticos para resolvê-lo.

7 Referências Bibliográficas

Adiri, I., Bruno, J., Frostig, E., & Rinnooy Kan, A. H. G. (1989). Single machine flow-time scheduling with a single breakdown. Acta Informatica, 26, 679–696.

Aggoune, R. (2004). Minimizing the makespan for the flow shop scheduling problem with availability constraints. European Journal of Operational Research, 153, 534–543.

Aggoune, R., & Portmann, M.-C. (2006). Flow shop scheduling problem with limited machine availability: A heuristic approach. International Journal of Production Economics, 99, 4–15.

Allaoui, A., Artiba, A., Elmaghraby, S. E., & Riane, F. (2006). Scheduling of a two-machine flowshop with availability constraints on the first machine. International Journal of Production Economics, 99, 16–27.

Armentano, V. A., Ronconi, D. P. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. Computers & Operations Research, 26(3), 219–235.

Baker, K. R.; Bertrand, J. W. M (1982). A dynamic priority rule for scheduling against due-dates. Journal of Operations Management, v. 3, p. 37-42.

Błazewicz, J., Breit, J., Formanowicz, P., Kubiak, W., & Schmidt, G. (2001). Heuristic _ algorithms for the two-machine flowshop with limited machine availability. Omega, 29, 599–608.

Breit, J. (2004). An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint. Information Processing Letters, 90, 273–278

Breit, J. (2006). A polynomial-time approximation scheme for the two-machine flow shop scheduling problem with an availability constraint. Computers & Operations Research, 33, 2143–2153.

Breit, J. (2007). Improved approximation for non-preemptive single machine flowtime scheduling with an availability constraint. European Journal of Operational Research, 183, 516–524.

Chang, S. Y., & Hwang, H.-C. (1999). The worst-case analysis of the MULTIFIT algorithm for scheduling non-simultaneous parallel machines. Discrete Applied Mathematics, 92, 135–147.

Cheng, T. C. E., & Wang, G. (2000). An improved heuristic for two-machine flowshop scheduling with an availability constraint. Operations Research Letters, 26, 223–229.

Du, J. , & Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research, 15 , 483-495 .

Fernandez-Viagas, V., Valente, J., & Framinan, J. (2018). Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. Expert Systems with Applications, 94 , 58–69 .

Framinan, J. , & Leisten, R. (2008). Total tardiness minimization in permutation flow shops: A simple approach based on a variable greedy algorithm. International Journal of Production Research, 46 (22), 6479–6498 .

Hasija, S. , & Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. International Journal of Production Research, 42 (11), 2289–2301

He, Y., Zhong, W., & Gu, H. (2006). Improved algorithms for two single machine Scheduling problems. Theoretical Computer Science, 363, 257–265.

Ho, J.C., & Wong, J.S. (1995). Makespan minimization for m parallel identical processors, Naval Research Logistics 42 935–948.

Hwang, H.-C., & Chang, S. Y. (1998). Parallel machines scheduling with machine shutdowns. Computers and Mathematics with Applications, 36, 21–31.

Hwang, H.-C., Lee, K., & Chang, S. Y. (2005). The effect of machine availability on the worst-case performance of LPT. Discrete Applied Mathematics, 148, 49–61.

Johnson, S.M. (1954), Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics, 1: 61-68

Karabulut, K. (2016). A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. Computers and Industrial Engineering, 98 , 300–307.

Kellerer, H. (1998). Algorithms for multiprocessor scheduling with machine release times. IIE Transactions, 30, 991–999

Kim, Y.-D. , 1993. Heuristics for flowshop scheduling problems minimizing mean tardiness. Journal of the Operational Research Society. 44 (1), 19–28 .

Kubiak, W., Błazewicz, J., Formanowicz, P., Breit, J., & Schmidt, G. (2002). Two-machine flow shops with limited machine availability. European Journal of Operational Research, 136, 528–540.

Kubzin, M. A., Potts, C. N., & Strusevich, V. A. (2009). Approximation results for flow shop scheduling problems with machine availability constraints. Computers & Operations Research, 36, 379–390.

Lee, C. Y. (1991). Parallel machine scheduling with non-simultaneous machine available time. Discrete Applied Mathematics, 30, 53–61.

Lee, C. Y. (1996). Machine scheduling with an availability constraints. Journal of Global Optimization, 9, 363–382.

Lee, C. Y. (1997). Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint. Operations Research Letters, 20, 129–139.

Lee, C. Y. (1999). Two-machine flowshop scheduling with availability constraints. European Journal of Operational Research, 114, 420–429.

Lee, C. Y., & Liman, S. D. (1992). Single machine flow-time scheduling with scheduled maintenance. Acta Informatica, 29, 375–382.

Lee, C. Y., & Liman, S. D. (1993). Capacitated two-parallel machines scheduling to minimize sum of job completion times. Discrete Applied Mathematics, 41, 211–222.

Liao, C.-J., Shyur, D.-L., & Lin, C.-H. (2005). Makespan minimization for two parallel machines with an availability constraint. European Journal of Operational Research, 160, 445–456.

Lin, G., He, Y., Yao, Y., & Lu, H. (1997). Exact bounds of the modified LPT algorithm applying to parallel machines scheduling with nonsimultaneous machine available times. Applied Mathematics: A Journal of Chinese Universities, 12, 109–116.

Lin, C.-H., & Liao, C.-J. (2007). Makespan minimization for two parallel machines with an unavailable period on each machine. The International Journal of Advanced Manufacturing Technology, 33, 1024–1030.

Lu, L., & Posner, M. E. (1993). An NP-hard open shop scheduling problem with polynomial average time complexity. Mathematical Methods of Operations Research, 18, 12–38.

Ma, Y., Chu, C., & Zuo, C. (2010). A survey of scheduling with deterministic machine availability constraints. Computers and Industrial Engineering, 58(2), 199-211

Nawaz, J., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for three-machine, n-job sequencing problem. Omega, 11, 91–95.

Ng, C. T., & Kovalyov, M. Y. (2004). An FPTAS for scheduling a two-machine flowshop with one unavailability interval. Naval Research Logistics, 51, 307–315.

Pan, J. C.-H., Chen, J.-S., & Chao, C.-M. (2002). Minimizing tardiness in a two-machine flow-shop.Computers & Operations Research, 29(7), 869–885

Panwalkar, S., Smith, M. & Seidmann, A. (1982). Common due date assignment to minimize total penalty for the one machine scheduling problem. Operations Research 30(2), 391-399

Pinedo, M. (2012). Scheduling: Theory, Algorithms and Systems. 4. ed. New York: Springer

Raman, N. (1995) Minimum tardiness scheduling in flow shops: Construction and evaluation of alternative solution approaches. Journal of Operations Management 12(2), 131-151

Rodammer, F.A. & White, Jr. K.P. (1988) A Recent Survey Of Production Scheduling. IEEE Transaction on Systems, Man and Cybernetics. 18(6), 841-851

Sadfi, C., Penz, B., Rapine, C., Błazewicz, J., & Formanowicz, P. (2005). An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. European Journal of Operational Research, 161, 3–10.

Schmidt, G. (2000). Scheduling with limited machine availability. European Journal of Operational Research, 121, 1–15

Sen, T. , & Gupta, S.K. , 1984. A state-of-art survey of static scheduling research involving due dates. Omega 12 (1), 63–76 .Wang, G., & Cheng, T. C. E. (2007a). An approximation scheme for two-machine flowshop scheduling with setup times and an availability constraint. Computers & Operations Research, 34, 2894–2901.

Vallada, E. , & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. OMEGA, The International Journal of Management Science, 38 (1–2), 57–67 .

Vallada, E. , Ruiz, R. , & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. Computers & Operations Research, 35 (4), 1350–1373 .

Wang, G., & Cheng, T. C. E. (2007b). Heuristics for two-machine flowshop scheduling with setup times and an availability constraint. Computers & Operations Research, 34, 152–162.

Wilson, J.M.: Alternative formulations of a flow-shop Scheduling problem. Journal of the Operational Research Society, 40(4), 395–399 (1989)

Xu, Z. , Xu, D., He, J., Wang, Q., Liu, A., Xiao, J. (2018). Mixed Integer Programming Formulations for Two-Machine Flow Shop Scheduling with an Availability Constraint. Arabian Journal for Science and Engineering, 43(2), 777–788.

Apêndice A – Modelagem do Problema em Python com o *solver Gurobi*.

```
import gurobipy as gp
from gurobipy import GRB
import pandas as pd
import time
def otimiza_gurobi(jobs,p,d,s1,s2,Major):
    # Create Model
    m = gp.Model("teste1")
    m.Params.TIME_LIMIT = 1800 # Limita o tempo de execução para 30 minutos

    # CREATE VARIABLES
    X = []
    for i in jobs:
        Xi = []
        for j in jobs:
            Xij = m.addVar(vtype=GRB.BINARY, name="x"+str(i)+ "," +str(j))
            Xi.append(Xij)

        X.append(Xi)

    S = {}
    for r in [1,2]:
        Sr = []
        for j in jobs:
            Srj = m.addVar(vtype=GRB.CONTINUOUS, name="S"+str(r)+ "," +str(j))
            Sr.append(Srj)
```

```
    S[r] = Sr

    m.update()

T = []
for j in jobs:
    Tj = m.addVar(vtype=GRB.CONTINUOUS, name="T"+str(j))
    T.append(Tj)

m.update()

# CREATE CONSTRAINTS

# Função Objetivo (01)
obj = 0
for Tj in T:
    obj = obj + Tj
m.setObjective(obj, GRB.MINIMIZE)

#Constraint (02)
for i in jobs:
    left = 0
    for j in jobs:
        left = left + X[i][j]
    m.addConstr(left == 1, ("c02-" + str(i)))

#Constraint (03)
for j in jobs:
    left = 0
```

```
    for i in jobs:
        left = left + X[i][j]
    m.addConstr(left == 1, ("c03-" + str(j)))

#Constraint (04)
for j in jobs:
     m.addConstr(S[1][j] - s1 <= Major - (Major * X[-2][j]), ("c04-" + str(j)))

#Constraint (05)
for j in jobs:
     m.addConstr(S[1][j] - s1 >= - Major + (Major * X[-2][j]), ("c05-" + str(j)))

#Constraint (04b)
for j in jobs:
     m.addConstr(S[2][j] - s2 <= Major - (Major * X[-1][j]), ("c04b-" + str(j)))

#Constraint (05b)
for j in jobs:
     m.addConstr(S[2][j] - s2 >= - Major + (Major * X[-1][j]), ("c05b-" + str(j)))

#Constraint (06)
for r in [1,2]:
    for j in jobs[:-1] :
        right = S[r][j]
        for i in jobs:
            right = right + p[r][i] * X[i][j]

        m.addConstr(S[r][j+1] >= right, ("c06-" +str(r)+ "," +str(j) ))
```

```
#Constraint (07)

for j in jobs:
    right = S[1][j]
    for i in jobs:
        right = right + p[1][i] * X[i][j]

    m.addConstr(S[2][j] >= right, ("c07-" + str(j) ))

#Constraint (08)
m.addConstr(S[1][1] >= 0, ("c08"))

#Constraint (09)
for j in jobs:
    right = S[2][j]
    for i in jobs:
        right = right + p[2][i] * X[i][j]  - d[i] * X[i][j]

    m.addConstr(T[j] >= right, ("c09-" + str(j)))

#Constraint (10)
for j in jobs:
    m.addConstr(T[j] >= 0, ("c10-" + str(j)))

m.update()

# OPTIMIZE

t0 = time.time()
```

```
m.optimize()

t1 = time.time()

time_execution = t1-t0

#print(round(m.objVal))

return round(m.objVal), time_execution
```

Apêndice B – Código das Heurísticas EDD, NEH-T e NEH-H

```
import math

import numpy as np

from time import process_time

from time import process_time_ns

import time

import random

#Obs: Nessa implementação a M1 é representada pelo índice 0. Nas insts 1 a 3 estava pelo índice 1.


class Job:

    def __init__(self,i,p,d):

        self.i=i     # número do job

        self.p=p     # Lista dos tempos de processamento

        self.d=d     # Due date

        self.S=[0,0] # lista dos instantes de início em cada máquina

        self.C=[0,0]   # lista dos instantes de conclusão em cada máquina

        self.T= 0  # Atraso do job


def calculaJob (job, S, C):

    job.S = S

    job.C = C

    job.T = max(job.C[1] - job.d, 0)


def calculaCmax (lista_jobs, s1=0, t1=0, s2=0, t2=0):

    m=2

    n = len(lista_jobs)

    s=[[0 for i in range(m)] for j in range(n)]
```

```
c=[[0 for i in range(m)] for j in range(n)]
p=[job.p for job in lista_jobs]

indisp1 = False # Indica se o período de indisponibilidade já ocorreu na M1
indisp2 = False # Indica se o período de indisponibilidade já ocorreu na M2

# Primeiro Job
  # Primeira Máquina
if p[0][0] <= s1:
    s[0][0] = 0
    c[0][0]=p[0][0]
else:
    indisp1 = True
    s[0][0] = t1
    c[0][0]= s[0][0] + p[0][0]

  # Próxima Máquina
if c[0][0] + p[0][1] <= s2:
    s[0][1] = c[0][0]
    c[0][1] = s[0][1] + p[0][1]
else:
    indisp2 = True
    s[0][1] = max(c[0][0],t2)
    c[0][1]= s[0][1] + p[0][1]

# Próximos Jobs
    #Primeira Máquina
for j in range(1,n):
    if (c[j-1][0] + p[j][0] <= s1) or indisp1:
```

```
            s[j][0] = c[j-1][0]
            c[j][0] = s[j][0] + p[j][0]
        else:
            indisp1 = True
            s[j][0] = max(c[j-1][0],t1)
            c[j][0] = s[j][0] + p[j][0]

        #Próxima Máquina
        possible_start_m2 = max(c[j-1][1], c[j][0]) # max(previous job finished on M2, this job finished on M1)
        if (possible_start_m2 + p[j][1] <= s2) or indisp2:
            s[j][1] = possible_start_m2
            c[j][1] = s[j][1] + p[j][1]

        else:
            indisp2 = True
            s[j][1] = max(possible_start_m2, t2)
            c[j][1] = s[j][1] + p[j][1]

    #print(s)
    #print(c)
    #print(p)

    total_tardiness = 0
    for job_i in range(len(lista_jobs)):
        calculaJob(lista_jobs[job_i], [s[job_i][0], s[job_i][1]], [c[job_i][0], c[job_i][1]])
        total_tardiness += lista_jobs[job_i].T

    Cmax = lista_jobs[-1].C[1]
```

```
    return Cmax, total_tardiness


def limite_inferior_atraso(job):
    return job.d - job.p[0] - job.p[1]


def ordena_jobs(lista_jobs):
    lista_jobs.sort(key = limite_inferior_atraso)
    return lista_jobs


def NEH(lista_jobs, s1=0, t1=0, s2=0, t2=0, criterio_insercao = 'T'):


    m = 2
    n = len(lista_jobs)
    bestSeq = []
    jobs_ordenados = lista_jobs.copy()
    jobs_ordenados = ordena_jobs(jobs_ordenados) # Ordena lista por limite inferior de
atraso


    # Adicionar primeiro job
    bestSeq.append(jobs_ordenados.pop(0))


    # Adicionar próximos jobs


    while len(jobs_ordenados) > 0:
        job_a_adicionar = jobs_ordenados.pop(0)


        lista_testes = bestSeq.copy()
```

```
melhorCmax = float('inf')
melhorT = float('inf')

for i in range(len(bestSeq) + 1):

    lista_testes.insert(i, job_a_adicionar)
    Cmax, T = calculaCmax(lista_testes, s1, t1, s2, t2)

    if criterio_insercao == 'T-Cmax':
            if T < melhorT:
                index_melhor = i
                melhorT = T
                melhorCmax = Cmax

            elif (T == melhorT) and (Cmax < melhorCmax):
                    index_melhor = i
                    melhorT = T
                    melhorCmax = Cmax

    else: # Critério T

        if T < melhorT:
            index_melhor = i
            melhorT = T


    lista_testes.pop(i)
```

```
        bestSeq.insert(index_melhor, job_a_adicionar) # Adiciona o job na melhor posição (em empate, ele fica na posição mais no começo da seq)




    Cmax, T = calculaCmax(bestSeq, s1, t1, s2, t2)


    #print(T)


    return [bestSeq, melhorT]


def csv_NEH(file_name, criterio_insercao = 'T'):


    df = pd.read_csv(file_name, sep = ';', header = 0)
    s1 = df.iloc[df.index[-1]]['num_job']
    t1 = df.iloc[df.index[-1]]['p1']
    s2 = df.iloc[df.index[-1]]['p2']
    t2 = df.iloc[df.index[-1]]['d']


    df_jobs = df[:-2]


    lista_jobs = []
    for index, row in df_jobs.iterrows():
        lista_jobs.append(Job(i= row.num_job,
                              p= [row.p1, row.p2],
                              d= row.d))


    del df
    del df_jobs
```

```
    #t0 = time.time() # Tempo real, não consegui com o tempo de execução

    t0 = time.process_time()

    lista_resultados = NEH(lista_jobs, s1, t1, s2, t2, criterio_insercao)

    #t1 = time.time()

    t1 = time.process_time()

    time_execution = t1-t0

    bestSeq = lista_resultados[0]
    melhorT = lista_resultados[1]

    return bestSeq, melhorT, time_execution

def ordena_jobs_EDD(lista_jobs):
    lista_jobs.sort(key = lambda job: job.d)
    return lista_jobs


def csv_EDD(file_name):

    df = pd.read_csv(file_name, sep = ';', header = 0)
    s1 = df.iloc[df.index[-1]]['num_job']
    t1 = df.iloc[df.index[-1]]['p1']
    s2 = df.iloc[df.index[-1]]['p2']
```

```
t2 = df.iloc[df.index[-1]]['d']

df_jobs = df[:-2]

lista_jobs = []
for index, row in df_jobs.iterrows():
    lista_jobs.append(Job(i= row.num_job,
                          p= [row.p1, row.p2],
                          d= row.d))

del df
del df_jobs

t0 = time.process_time()

lista_resultados = ordena_jobs_EDD(lista_jobs)

t1 = time.process_time()

time_execution = t1-t0

Cmax, T = calculaCmax(lista_resultados, s1, t1, s2, t2)
bestSeq = lista_resultados
melhorT = T

return bestSeq, melhorT, time_execution
```