BRUNO COELHO DE OLIVEIRA

MINIMIZAÇÃO DO CONSUMO DE ENERGIA E ATRASO TOTAL EM UM AMBIENTE *FLOW SHOP* COM RESTRIÇÕES DE DISPONIBILIDADE

São Paulo
2024

BRUNO COELHO DE OLIVEIRA

MINIMIZAÇÃO DO CONSUMO DE ENERGIA E ATRASO TOTAL EM UM AMBIENTE *FLOW SHOP* COM RESTRIÇÕES DE DISPONIBILIDADE

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

São Paulo
2024

BRUNO COELHO DE OLIVEIRA

MINIMIZAÇÃO DO CONSUMO DE ENERGIA E ATRASO TOTAL EM UM 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: Professora Dra. Débora Pretti Ronconi

São Paulo

2024

Catalogação-na-publicação

Oliveira, Bruno Coelho

Minimização do consumo de energia e atraso total em um ambiente *flow shop* com restrições de disponibilidade/ B.C. Oliveira -- São Paulo, 2024.

86 p.

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

1. Programação da Produção. 2. Consumo de energia 3. Flow shop. 4. Restrição de disponibilidade. 5. Multi-objetivo. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Produção II.t.

Aos que sempre me apoiaram

AGRADECIMENTOS

Primeiramente gostaria de agradecer aos meus pais, Simone e Eduardo, e à minha irmã, Laura, pelo apoio e suporte que sempre me deram em todos os aspectos da vida, presentes tanto em momentos bons quanto ruins. Tenho certeza que vocês têm um papel fundamental em tudo que conquistei e irei conquistar. Aos demais familiares, obrigado pelo carinho e pela base que me proporcionam desde sempre.

A todos amigos que conheci durante a jornada da graduação, pessoas que fizeram esses anos mais leves e agradáveis. Especialmente Aline, Gabriela e Leonardo, pela parceria e amizade desde os primeiros dias de aula.

Ao time de futsal da Poli, onde pude conhecer grandes amigos e viver bons momentos que estarão na minha memória para sempre.

A todos funcionários e corpo docente da Escola Politécnica que permitiram o meu desenvolvimento enquanto aluno e pessoa durante todos esses anos, sempre entregando um ótimo serviço e educação de qualidade.

À professora Débora Pretti Ronconi pelo compromisso com os alunos, e principalmente pelo auxílio e orientação durante a elaboração deste trabalho, sempre se colocando disposta e solicita para contribuir com a minha evolução.

“O futuro depende do que fazemos no presente.”

Mahatma Gandhi

RESUMO

Com o aumento da relevância de pautas ambientais, o gerenciamento da produção com um olhar ecológico tem ganhado cada vez mais visibilidade. Este trabalho tem como objetivo abordar o problema de minimização de consumo total de energia e atraso total em um ambiente *flow shop* com restrições de disponibilidade, onde tarefas devem ser processadas sequencialmente em todas as máquinas seguindo uma mesma ordem. A estratégia de ligar e desligar máquinas entre períodos ociosos é considerada para o controle de energia. Para este problema foi proposto um modelo matemático multiobjetivo utilizando a abordagem lexicográfica, onde o consumo de energia é o principal objetivo a ser otimizado e o atraso total o objetivo secundário. Devido ao caráter *NP-hard* deste problema também foram propostos métodos heurísticos, sendo eles uma heurística construtiva baseada na regra de despacho FPD (*fitting processing times and due dates*) e uma heurística de melhoria que utiliza buscas locais e perturbação para explorar vizinhanças da solução obtida pela heurística construtiva. O modelo matemático e os métodos heurísticos foram validados e testados em 45 instâncias geradas com diferentes tamanhos baseados na literatura. Os resultados obtidos demonstram que a resolução do problema através do modelo matemático apresenta dificuldades para resolução de instâncias de maiores escalas, desta forma sendo necessário o uso das heurísticas, que demonstraram alcançar resultados satisfatórios.

Palavras-chave: Programação da produção. Consumo de energia. *Flow shop*. Restrição de disponibilidade. Multi-objetivo

ABSTRACT

With the increasing relevance of environmental issues, production management with an ecological perspective has gained greater visibility. This study aims to address the problem of minimizing total energy consumption and total tardiness in a flow shop environment with availability constraints, where tasks must be processed sequentially on all machines following the same order. The strategy of turning machines on and off during idle periods is considered for energy control. For this problem, a multi-objective mathematical model was proposed using the lexicographic approach, where energy consumption is the primary objective to be optimized, and total tardiness is the secondary objective. Due to the NP-hard nature of this problem, heuristic methods were also proposed, including a constructive heuristic based on the FPD (Fitting Processing Times and Due Dates) dispatching rule and an improvement heuristic that uses local searches and perturbation to explore the neighborhoods of the solution obtained by the constructive heuristic. The mathematical model and heuristic methods were validated and tested on 45 instances generated with different sizes based on the literature. The results demonstrate that solving the problem using the mathematical model faces challenges with larger-scale instances, making the use of heuristics necessary. The heuristics showed the ability to achieve satisfactory results.

Keywords: Production scheduling. Energy consumption. Flow shop. Availability constraint. Multi-objective.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A programação da produção, ou *Scheduling*, é um processo de tomada de decisão que é utilizado como a base de muitas indústrias de manufatura e serviços. Ela lida com a alocação de recursos a tarefas em um dado período de tempo com objetivo de otimizar um ou mais objetivos (Pinedo, 2016). Estes recursos e tarefas podem ser visto de diferentes formas, como equipes para realização de um projeto, pistas para o pouso de aviões, máquinas para produzir um produto, entre outras. Quando nos referimos à manufatura, é muito comum chamarmos os recursos de máquinas e tarefas de *jobs*.

Dentro deste contexto da programação da produção, podemos ter diversas variações que vão moldar como o problema é, e também como pode ser resolvido, dentro dessas características temos a configuração de máquinas, ou seja, máquinas únicas, máquinas em paralelo, *flow shop* e *job shop*. No caso do *flow shop* existem *m* máquinas em série e cada tarefa tem que ser processada em cada uma das *m* máquinas, seguindo o mesmo roteiro, ou seja, passar pela máquina 1, depois pela máquina 2, até a máquina *m*. Estes problemas de *flow shop* muitas vezes possuem restrições que vão proibir uma tarefa de passar na frente de outra na fila para ser processada nas máquinas (primeira que entra é a primeira que sai, *FIFO*) e assim caracterizando o que é chamado de *flow shop* permutacional.

Dentro desses casos de *scheduling*, modelar estes problemas contando que as máquinas estão sempre disponíveis e operando não representa bem o que acontece em indústrias e casos reais, desta forma surgindo o problema de disponibilidades nas máquinas. As indisponibilidades podem representar situações como manutenções programadas para uma dada máquina, por exemplo, e é essencial que ao programarmos a tarefa seja levado em consideração essas possíveis restrições quanto ao uso dos recursos.

Sempre que fazemos a programação de tarefas temos um objetivo a ser alcançado, um dos objetivos mais comuns de se encontrar é o *makespan*, que diz respeito ao tempo total para se realizar todas as tarefas, mas um outro critério importante é o de atraso total, pois quando uma tarefa não é completada dentro do tempo esperado existem alguns custos referentes a este atraso para a empresa (Armentano e Ronconi, 1999), desta forma é interessante organizar as tarefas de forma que este atraso total seja o menor possível.

Outro ponto que vem tomando cada vez mais força na sociedade em geral são questões de sustentabilidade e na programação da produção não é diferente, empresas de

manufatura tem dado cada vez mais importância a sustentabilidade por diversos fatores: consciência ambiental, a diminuição dos recursos não-renováveis, legislações mais rigorosas, preferência do consumidor em produtos sustentáveis, entre outros (Giret, 2015). Esta visão sustentável da programação da produção está diretamente ligada ao aumento da eficiência energética, sendo um critério para ser levado em conta na otimização de *scheduling*.

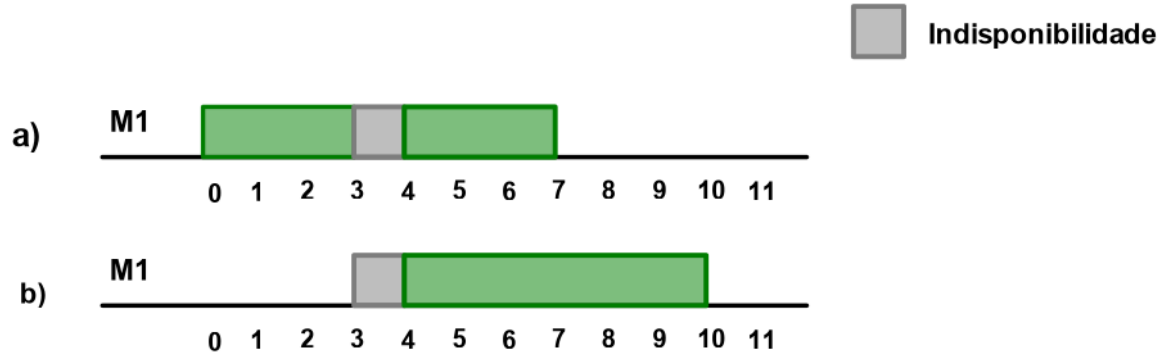
Pouco se encontra na literatura sobre problemas que abordam a programação da produção em *flow shops* com restrições de disponibilidade e que tenham como objetivo otimizar a eficiência energética e tempo total de atraso, desta forma o presente trabalho busca estudar este problema, para isso será proposto um modelo matemático para resolvê-lo, assim como a otimização do mesmo. Pelo fato do problema de minimização de atraso total ser *NP-hard,* muitas vezes a solução exata pode não ser a mais viável, então para a resolução deste problema será proposto também a utilização de métodos não-exatos, e posteriormente a partir dos testes computacionais fazer a comparação e validação entre métodos de solução.

2. DESCRIÇÃO DO PROBLEMA

O problema a ser resolvido tem como base a otimização da programação da produção em ambientes *flow shops* para m máquinas, mais especificamente o *flow shop* permutacional, desta forma além das tarefas terem que passar por todas as máquinas, a ordem em que as mesmas são executadas em cada uma das m máquinas deve ser a mesma.

Além disso o problema também leva em consideração restrições de disponibilidades nas máquinas, em que cada uma delas pode apresentar janelas de indisponibilidades, a partir disso teremos dois cenários que são caracterizados segundo a definição de *resumable* e *non-resumable* feita por Lee (1996) a qual nos traz que uma janela de indisponibilidade pode ser caracterizada como *resumable* se caso uma tarefa não consiga ser terminada antes do tempo de indisponibilidade, ela possa retomar seu processamento do momento em que parou, assim que a máquina estiver disponível novamente. Já uma janela *non-resumable*, as tarefas devem ser reiniciadas se não for possível terminá-las antes do período de indisponibilidade. Neste trabalho será abordado janelas *non-resumables*.

Figura 1: Exemplo de alocação de uma tarefa em uma máquina com janelas de indisponibilidade *resumable* (a) e *non-resumable* (b)



Fonte: Elaborado pelo autor

Para melhor entendermos este conceito, podemos observar na Figura 1 a programação de uma única tarefa com tempo de processamento de 6 unidades de tempo em uma máquina que apresenta um período de indisponibilidade entre os instantes 3 e 4 , no cenário (a) temos o caso *resumable* no qual a tarefa começa a ser realizada no instante 0 e quando chega o período de indisponibilidade ela é pausada para que assim que a máquina esteja disponível retome o seu processamento de onde parou, finalizando assim no instante 7. Já no cenário (b) temos um caso *non-resumable* em que por não ser possível de ser realizada por completo a

tarefa devido o tempo de indisponibilidade ela começa apenas no instante 4, quando vai poder ser realizada até sua finalização sem interrupções.

Existem diferentes maneiras de se considerar essas restrições quanto a sua natureza na programação da produção, dois tipos de suposições são consideradas pelos pesquisadores, na primeira, os tempos de indisponibilidade são conhecidos e fixados previamente (Aggoune, 2006), na segunda, esses tempos de indisponibilidade são flexíveis e podem ser planejados (Cui *et al.*, 2016). Para o primeiro tipo (fixos), podemos considerar que a indisponibilidade vai acontecer apenas uma vez em cada máquina ou seguindo intervalos que não se alteram. Já no segundo tipo (flexível) podemos ter, por exemplo, que o tempo que uma máquina processa continuamente tarefas não pode extrapolar um dado valor sem que não haja um período de indisponibilidade, que pode ser visto como um período de manutenção. Consideramos para este trabalho o primeiro tipo, onde os períodos de indisponibilidade já são conhecidos e fixados previamente.

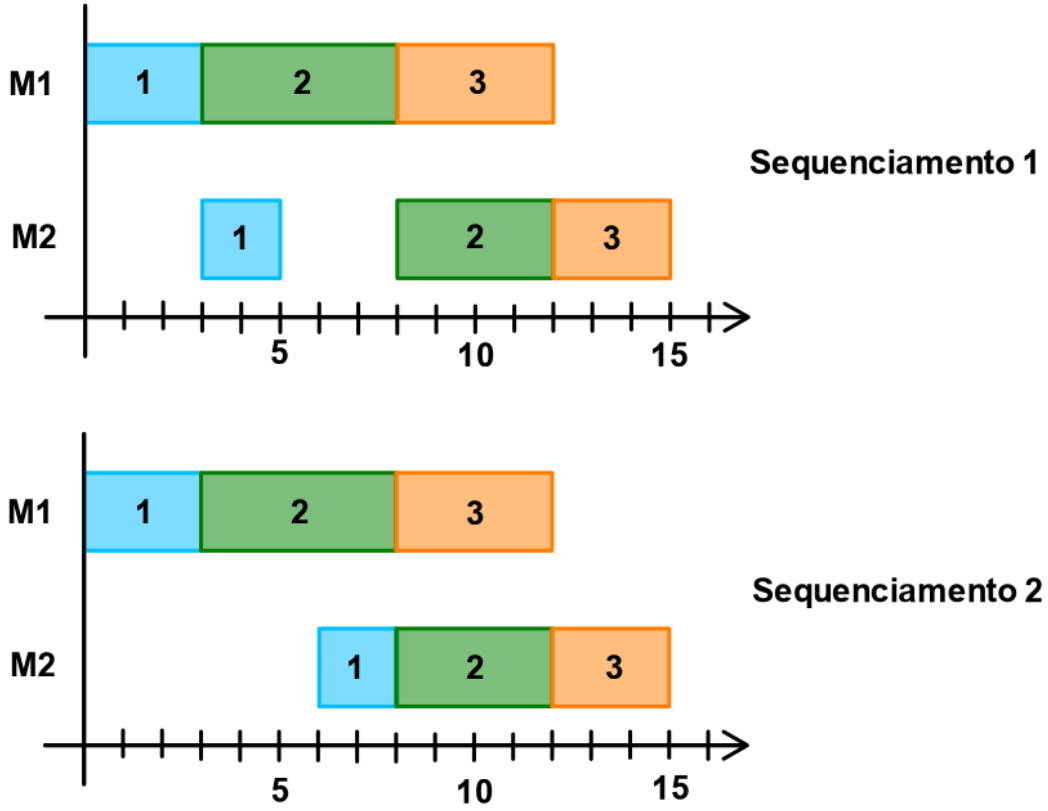
Outro ponto para definir o problema é quanto ao objetivo de otimização, o presente trabalho tem como proposta trazer uma perspectiva tanto sustentável quanto econômica para este problema de *scheduling*, desta forma ao realizarmos a modelagem matemática, será utilizado a abordagem multi-objetivo, mais especificamente a otimização lexicográfica. Quando falamos de otimização multi-objetivo muitas vezes podemos ter objetivos que são conflitantes entre si e portanto quando melhoramos um objetivo podemos pagar com a piora de outro. Zhang *et al.* (2020) coloca três principais métodos que são adotados para lidar com problemas multi-objetivo: ponderado, lexicográfico e fronteira de pareto. Cada método tem suas vantagens e desvantagens e sendo assim a escolha entre eles vai depender da preferência do tomador de decisão (Coello, 1999). Na abordagem ponderada são atribuídos pesos para os diferentes objetivos, determinando a relação entre eles e assim a solução ótima é encontrada por estes ajustes de pesos. A fronteira de Pareto representa o conjunto de soluções onde melhorar um objetivo só é possível piorando outro, sem preferências de peso entre eles, e assim o tomador de decisão pode escolher a solução que melhor se encaixa na suas preferências

Já a abordagem lexicográfica, a qual será utilizada neste trabalho, objetivos de otimização são colocados em ordem de prioridade, neste tipo de otimização o tomador de decisão está disposto a aceitar uma solução sub-ótima para os objetivos menos críticos para alcançar uma solução ótima para o critério mais importante, cada objetivo é otimizado

enquanto os outros objetivos são fixados no nível desejado (Isermann, 1982). Neste trabalho o critério principal a ser otimizado será o de eficiência energética, em que se busca diminuir a energia gasta pelas máquinas para a realização das tarefas programadas, este objetivo é considerado não regular. Segundo Pinedo (2016) um objetivo pode ser chamado de regular se o mesmo é não-decrescente em relação ao *completion time* das tarefas, isto é, à medida que o tempo de término das tarefas aumenta, o valor da função objetivo cresce de maneira previsível ou se mantém. Em problemas de minimização de energia, o objetivo frequentemente envolve reduzir o consumo total, que pode variar dependendo do momento em que as tarefas são executadas, fazendo com que a função objetivo possa ser mais complexa e envolver penalidades e variáveis que irão fazer com que o valor da função objetivo possa diminuir à medida que o tempo de término das tarefas aumenta.

Na Figura 2 podemos ver exemplo que ilustra dois diferentes sequenciamentos de três tarefas em duas máquinas em um ambiente *flow shop* permutacional, suponha que cada unidade de tempo que as tarefas levam para ser processadas consomem uma unidade de energia e cada unidade de tempo de período ocioso entre tarefas, isto é, períodos em que a máquina está ligada mas não processa nenhuma tarefa, consomem também uma unidade de energia. Vamos considerar o consumo de energia como a soma de energia gasta processando e energia gasta em tempo ocioso. No sequenciamento 1 temos um consumo de energia total de 24 unidades de energia, sendo 21 unidades gastas com o processamento de tarefas e 3 unidades devido ao tempo ocioso entre tarefas. Já no sequenciamento 2 não possuímos tempo ocioso entre tarefas, portanto o consumo de energia total é de 21 unidades, desta forma mesmo com o aumento do tempo de conclusão da tarefa 1, que passou do instante 5 para o instante 8, a função objetivo decresceu, o que caracteriza um objetivo não regular.

Figura 2: Sequenciamentos de três tarefas em duas máquinas em um ambiente *flow shop* permutacional



Fonte: Elaborado pelo autor

O segundo critério de otimização é o de atraso total, que visa minimizar a soma de atrasos na entrega de todas as tarefas, este critério por sua vez se encaixa na definição objetivos regulares, sendo diretamente relacionada com o tempo de conclusão das tarefas, assim a função objetivo tende a aumentar de maneira previsível conforme as tarefas se afastam dos prazos.

Há diversas maneiras de fazermos o controle do consumo energético pelas máquinas de um ambiente de produção, um dos métodos encontrados na literatura é o de desligar e ligar as máquinas para que as mesmas não gastem energia desnecessariamente (Mouzon *et al.*, 2007), operando no chamado de *IDLE time,* que significa o tempo ocioso do sistema, em que a máquina está ligada mas não está processando nenhuma tarefa. Outra maneira que pode ser encontrada na literatura é o controle da velocidade das máquinas (Mansouri *et al.*, 2016), onde é atribuído diferentes velocidades de processamento, geralmente representando uma velocidade lenta, normal e rápida, que irão influenciar diretamente no gasto de energia das máquinas, quanto mais rápido é o processamento de uma tarefa, maior é a energia gasta pela

mesma. Neste trabalho será utilizado a abordagem de desligar e ligar a máquina para o controle da energia gasta no processamento das tarefas.

Portanto o objetivo do presente trabalho é apresentar a modelagem de um *flow shop* com *m* máquinas, considerando restrições de disponibilidade *non-resumable*, onde os objetivos de otimização são a redução do consumo de energia, como objetivo principal, e o atraso total, como objetivo secundário, utilizando a estratégia de ligar e desligar máquinas no tempo ocioso. Após isso, por meio de instâncias geradas, validar o modelo resolvendo as instâncias através de um *solver* e também utilizar métodos não exatos para construir soluções para o problema, por meio de heurísticas, que se fazem necessárias devido a características *NP-hard* do problema.

3. REVISÃO DA LITERATURA

Nesta seção será apresentado o que se encontra a respeito dos assuntos abordados neste trabalho, ou seja, uma visão sobre a programação da produção, assim como as características que fazem parte do problema aqui estudado. Trazendo um maior aprofundamento em *flow shops*, restrições de disponibilidade, e como são aplicadas a este tipo de problema e depois uma revisão da eficiência energética aplicada na programação da produção.

Com a contribuição dos autores que aqui serão evidenciadas busca-se obter uma base teórica para a formulação matemática, assim como ferramentas para validar e solucionar o problema tratado neste trabalho.

3.1. Programação da produção

O problema da programação da produção é a organização da execução de um conjunto de tarefas com o passar do tempo, de forma a atingir um dado objetivo, alocando tarefas a recursos, este problema pode ser encontrado em diferentes situações como por exemplo unidades de processamento em um ambiente computacional. A programação da produção é de grande importância para garantir a eficiência de processos e a melhor utilização de um dado recurso, levando em consideração restrições e particularidades do mesmo. A partir da resolução deste problema é possível obter uma descrição da execução das tarefas, também chamadas de “jobs”, e a alocação de recursos, também chamados de “máquinas”, com o passar do tempo buscando otimizar um ou mais objetivos.

A programação da produção possui diferentes aspectos que levam a diferentes abordagens, por exemplo, uma delas diz respeito a modelos determinísticos, que não incorporaram aleatoriedade ou incerteza, já a outra de modelos estocásticos, que por sua vez incorporam processos probabilísticos que estão relacionados a incerteza. Desta forma, a partir das diferenças quanto à modelagem dos problemas é possível atender uma maior quantidade de cenários.

Segundo Pinedo (2016) um problema de programação da produção é definido por 3 características, o ambiente da máquina, as condições e restrições de processamento, e por fim o objetivo de otimização. Dentre os ambientes de máquina que podemos encontrar temos:

- Máquina única: O caso mais simples possível e é definido como um caso especial de todos os outros mais complexos, dificilmente representa um caso real.
- Máquinas paralelo: m máquinas em paralelo, em que um tarefa precisa de uma única operação a qual pode ser executada em qualquer uma das máquinas. Além deste temos algumas variações, em que em um deles máquinas são idênticas, e no outro elas podem assumir diferentes velocidades, que vão influenciar no tempo de processamento das tarefas.
- *Flow shop*: m máquinas em série, onde cada tarefa deve ser processada em cada uma das m máquinas, seguindo o mesmo roteiro, ou seja, devem passar pela máquina 1, depois na máquina 2, até a máquina m. Depois de executado em uma máquina, a tarefa entra na fila para ser executada na próxima seguindo alguma regra, que geralmente é a "primeiro que entra é o primeiro que sai" (FIFO), caracterizando assim o chamado *flow shop* permutacional.
- *job shop*: m máquinas das quais cada tarefa pode apresentar um roteiro diferente a ser seguido, desta forma podendo, ou não, ser necessário o processamento de uma tarefa por todas máquinas, e tendo a possibilidade de passar por uma mesma máquina mais de uma vez.

Outro diferente aspecto é o objetivo que o problema está abordando, alguns exemplos que podemos encontrar são:

- *Makespan*: mede o tempo que a última tarefa que saiu do sistema é completada, a minimização do mesmo implica numa boa eficiência das máquinas.
- Atraso máximo: mede a maior violação do tempo de entrega das tarefas que estão sendo processadas.
- Atraso total: mede a soma do atraso de todas as tarefas que foram processadas.

Desta forma, podemos encontrar diferentes combinações que irão influenciar na maneira que o problema é modelado, agora iremos entender mais detalhadamente o problema do *flow shop,* que é discutido neste trabalho.

3.2. *Flow shop*

O *flow shop*, como explicado anteriormente, é um tipo de ambiente de máquinas que podemos encontrar em problemas de *scheduling*, esta categoria possui uma grande relevância

de aplicação prática, e é amplamente estudada. A sua definição é dada como um sistema de processamento no qual uma sequência de operações de uma determinada tarefa é completamente especificada, e todas as tarefas visitam as unidades de trabalho levando em conta um mesmo roteiro.

Dentro desta classe ainda podemos ter diferentes tipos de características que vão influenciar como o problema é chamado e tratado, por exemplo podemos ter o *flow shop* permutacional em que as tarefas devem seguir a mesma ordem de processamento em todas as máquinas. E por outro lado podemos encontrar também os chamados *flow shops* não permutacionais, onde as tarefas podem ter diferentes sequências dentro de cada máquina. Tanto os problemas permutacionais quanto não permutacionais em sua grande maioria são modelados levando em conta que as tarefas podem ser armazenadas no momento que saem das máquina *r* até o momento que deve entrar na máquina *r+1*, porém podemos também encontrar problemas que irão tratar do caso em que não é possível ter esse armazenamento entre as máquinas e desta forma caracterizando o chamado *no-wait flow shop*, que se uma tarefa termina o seu processamento na máquina *r* mas a máquina *r+1* ainda não está livre, a tarefa impedirá que a máquina *r* possa ser utilizada, neste trabalho abordaremos o caso geral em que as tarefas podem ser armazenadas entre máquinas.

3.3. Restrições de disponibilidade

Modelar problemas de programação de tarefas supondo que as máquinas não terão tempos de indisponibilidade não representa bem o que acontece no mundo real, é comum termos situações de manutenções preventivas ou paradas programadas para que se mantenha a máquina em perfeitas condições para uso, ou até mesmo podemos lidar com situações de quebra e mau funcionamento que impedirão as máquinas de processar as tarefas em um dado intervalo de tempo. Devido a este fator para a adequação dos problemas de programação da produção é necessário levar em conta restrições quanto a disponibilidade das máquinas. Segundo a definição feita por Lee (1996), podemos ter os casos *resumables* e *non-resumables* que irão definir se após uma janela de indisponibilidade um tarefa pode continuar de onde parou, se for interrompida no meio do processamento (*resumable*), ou se não pode e deve ser reiniciada (*non-resumable*), posteriormente o caso *semiresumable* é definido por Lee (1999), que é caracterizado quando uma operação interrompida deve recomeçar parcialmente depois que a máquina estiver disponível novamente.

O caso *non-resumable* de restrição de disponibilidade para uma máquina foi estudado por Adiri *et al.* (1989) que foca na otimização da soma dos *completion time*, eles mostraram que o problema é *NP-hard* e que a sequência SPT (*Shortest processing time*), que corresponde a ordenar as tarefas em uma ordem não decrescente do seu tempo de processamento, tem uma margem de erro relativa de menos de ¼, quando levamos em conta uma única janela de indisponibilidade. Posteriormente, para o mesmo problema, um algoritmo de aproximação MSPT foi proposto por Sadfi etl al. (2005) e um algoritmo paramétrico O(nlog n) com melhores casos de piores margens de erros foi proposto por Breit (2007).

Para o caso *resumable* de restrição de disponibilidade para uma máquina Lee (1996) mostra que os problemas que tem foco na minimização do *makespan*, atraso máximo e soma dos *completion time*, podem ser resolvidos otimamente por uma sequência arbitrária, regra SPT e pela regra EDD (*Earliest Due Date*) respectivamente.

3.4. *Flow shop* com restrição de disponibilidade

De acordo com Ma, Chu, Zuo (2010) existem muitos estudos de *flow shops* com restrições de disponibilidade para duas máquinas, porém poucos para múltiplas máquinas, além disso a maioria dos estudos levam em consideração o *makespan* como o critério de otimização. O primeiro estudo de programação de tarefas em *flow shop* com restrições de disponibilidade (caso *resumable*) para duas máquinas foi realizado por Lee (1997), levando em conta o critério de otimização do *makespan*, neste estudo ele mostra que o problema se torna *NP-hard* para duas máquinas se tivermos apenas uma janela de indisponibilidade somente na primeira ou somente na segunda máquina, ele também mostra que o algoritmo de Johnson leva a uma margem de erro relativa menor ou igual a 1.

Para o caso *resumable* onde cada uma das máquinas pode ter um número arbitrário de janelas de indisponibilidade Błazewicz *et al*. (2001) apresenta duas heurísticas construtivas e uma heurística de busca local para resolver o problema, estas heurísticas foram testadas em problemas fáceis e mais difíceis de até 100 tarefas e 10 intervalos de indisponibilidade e obtiveram o pior desvio relativo do ótimo de 2,6% para os problemas mais fáceis e 44,4% para os difíceis. Kubiak *et al.* (2002) considera o problema com várias janelas de indisponibilidade somente na primeira máquina, somente na segunda ou em ambas, e mostram que o problema é *NP-hard* até para o caso que ocorre apenas em uma máquina e que se existir pelo menos uma janela na segunda máquina não existem heurísticas com tempo

polinomial que apresentem um erro relativo constante. Além disso Kubzin *et al.* (2009) apresenta um algoritmo de aproximação para o problema com múltiplas janelas de indisponibilidade na primeira máquina.

Para o caso *non-resumable* Allaoui *et al*. (2006) considera o problema com indisponibilidade apenas na primeira máquina, e propõem um modelo de programação dinâmica que é independente do tempo de processamento das tarefas. Lee & Kim (2017) estudam o problema da programação de tarefas em um *flow shop* de duas máquinas com objetivo de minimizar o atraso total, para o caso em que a máquina do primeiro estágio precisa de manutenção preventiva (restrição de disponibilidade), dado um período de tempo cumulativo entre a manutenção anterior. Este estudo considera o caso *non-resumable*, eles propõem um algoritmo de *branch and bound*, com limites inferiores definidos por programações parciais e limite superior definido por um algoritmo heurístico, como resultado obtiveram que o algoritmo consegue achar soluções ótimas para problemas de até 24 tarefas em um tempo computacional razoável.

Para múltiplas máquinas são poucos os estudos que podemos encontrar, Aggoune (2004) considera o problema com várias janelas de indisponibilidade (*non-resumable*) em cada máquina, devido o problema ser fortemente *NP-hard* ele propõem uma busca tabu e um algoritmo genético para resolvê-lo. Posteriormente Aggoune e Portmann (2006) apresentam uma abordagem geométrica temporizada para resolver o problema com duas tarefas e com base nisso uma heurística para resolver aproximadamente o problema com mais de duas tarefas. Outras pesquisas que consideram múltiplas máquinas comumente levam em consideração o *flow shop* híbrido, uma mistura do ambiente de máquinas em série e paralelo, onde temos uma série de estágios nos quais cada estágio possui múltiplas máquinas em paralelo, e as tarefas devem ser processados seguindo uma mesma ordem de estágios.

3.5. Eficiência energética na Programação da Produção

Um tópico que está muito em alta atualmente é a sustentabilidade, trazendo cada vez mais um olhar de responsabilidade para com o meio ambiente, isso se deve ao fato de desafios relacionados a crises de recursos, energia, mudanças climáticas e ambientais estarem cada vez mais comum, desta forma o mundo como um todo trabalha na estipulação de metas e objetivos para balancear aspectos econômicos, sociais e ambientais. A indústria tem um papel vital nestes objetivos visto o grande consumo de recursos e energia que as mesmas fazem utilização, por exemplo, a indústria de manufatura consumiu quase que um terço do

consumo energético global (Gao *et al*., 2019), além da grande quantidade de poluentes emitidos por estas. Para atingir esse desenvolvimento sustentável, podemos utilizar o chamado "*Green Shop Scheduling*", a programação da produção com um olhar sustentável, que tem potencial de aumentar significativamente a eficiência energética a um custo de praticamente zero (Li, 2022).

Os problemas de programação da produção verdes ou *green scheduling shop problems* (GSSPs) são extensões dos problemas clássicos de programação da produção, porém mais orientados aos recursos e ambiente, isto é, os problemas clássicos tendem a otimizar apenas indicadores econômicos, mas sem ter uma visão sobre o consumo energético e impacto ambiental. Estes problemas em sua grande maioria possuem abordagens multi-objetivos, tendo não só objetivos econômicos, como *makespan*, atraso total e custo de produção, por exemplo, mas também objetivos ambientais, que visam diminuir o consumo de energia e a poluição do ambiente, é muito comum observamos este consumo energético ou emissão de poluentes relacionados a duas estratégias na programação da produção, a primeira é a estratégia de ligar e desligar máquinas ociosas para reduzir a energia gasta pela mesma e a segunda estratégia é baseada na velocidade de processamento das máquinas, quanto menor for a velocidade, mais lento é o processamento da tarefa porém o consumo energético é reduzido.

Um dos primeiros trabalhos relevantes nessa área de economia energética foi o de Mouzon *et al*. (2007) que estudou o uso da programação da produção para minimização do uso energético em uma única máquina, baseado no conceito de desligar a máquina quando a mesma não precisava ser utilizada. Posteriormente Mouzon & Yildrim (2008) desenvolveram um *framework* para resolver o problema multi-objetivo que minimiza o consumo total de energia e o atraso total para uma máquina por meio do método GRASP (*Greedy Randomised Adaptive Search Procedure*) e constataram que a medida que o atraso total diminui o consumo energético aumenta.

O problema de *flow shop* permutacional verde ou *green permutation flow shop problem* (GPFSP), utilizando a abordagem baseada na velocidade de processamento das máquinas pode ser encontrado com diversas funções objetivo na literatura. Fang *et al.* (2013) consideraram este problema com uma restrição de pico de energia consumida, juntamente com funções objetivos baseadas em tempo, desenvolvendo um modelo matemático e abordagens para resolver este problema. Mansouri *et al*. (2016) abordou o GPFSP para duas

máquinas e analisou qual era a relação entre minimizar o *makespan* e a energia total consumida a partir de um modelo de programação linear inteira mista usado para obter a fronteira de Pareto dos dois objetivos de otimização.

Foumani & Smith-Miles (2019) estudam um problema multi-objetivo do *flow shop* verde para minimizar o *makespan* e a emissão total de carbono, eles propuseram um modelo de programação linear inteiro misto e usando a abordagem de agregar pesos aos objetivos transformaram o problema original em um problema de objetivo de otimização único.

3.6. Eficiência energética na Programação da Produção com restrições de disponibilidade

Pouco se encontra na literatura quanto aos problemas de GSSPs levando em consideração a restrição de disponibilidade de máquinas. Assia *et al*. (2019) aborda o *flow shop* não permutacional com intervalos de indisponibilidade e foco na minimização da energia total consumida e *makespan*, com intervalos de indisponibilidades periódicos fixos em um caso e no outro com intervalos flexíveis onde a máquina não pode superar um dado tempo de trabalho contínuo, em ambos os casos as tarefas são *non-resumables*. Eles apenas propõem modelos matemáticos de programação linear binária mista, sem se aprofundar em métodos e testes computacionais.

Cui & Lu (2020) já trazem uma abordagem bi-objetivo, com foco na minimização do *makespan* e consumo energético, para o problema do *flow shop* com restrições de pico de demanda energética. Eles propuseram um modelo integrado baseado no planejamento da produção, isto é, com planejamento de manutenção preventiva e controle de energia. Foi formulado um modelo matemático, assim como um algoritmo meta-heurístico baseado no algoritmo evolutivo NSGA - II, a partir de testes computacionais os autores puderam constatar que os impactos da manutenção preventiva e restrições de pico de demanda energética influenciam fortemente na sequência ótima das tarefas.

Ao conhecimento do autor não existem trabalhos que abordam o problema multi-objetivo de energia total consumida e tempo total de atraso para um *flow shop* permutacional levando em conta restrições de disponibilidade fixas.

4. MODELO MATEMÁTICO

Com a descrição do problema, podemos montar um modelo matemático com base nas ideias de Wilson (1989), que aborda diferentes modelos matemáticos para o *flow shop* e Meng *et al.* (2019), que apresenta seis diferentes modelos de programação inteira mista para o problema do *job shop* com objetivo de minimizar a energia total consumida. Assim, o modelo proposto neste trabalho, por meio de expressões matemáticas, irá restringir as soluções de modo que respeite as condições que foram colocadas na definição e otimize o sequenciamento das tarefas para atingir a melhor eficiência energética com olhar também para o atraso total, deste modo será necessário definir os parâmetros, variáveis de decisões e as equações que irão compor o modelo.

Neste exemplo são considerados *n* tarefas (caso *non-resumable*) a serem alocadas em *m* máquinas de forma que cada uma das máquinas pode apresentar uma janela de indisponibilidade, esse período de indisponibilidade na máquina *r* é visto como uma tarefa, de índice *n*+*r* a ser realizada. Desta forma a tarefa *n*+*r* deve possuir o instante de início igual ao instante em que a indisponibilidade na máquina *r* se inicia (s_r), tempo de processamento ($p_{r,\,n+r}$) somente na máquina em que a indisponibilidade ocorre e a data de entrega (d_{n+r}) deve ser um valor suficientemente grande, pois não levaremos em consideração atraso nesta tarefa.

4.1. Variáveis

$X_{i,j}$ Variável binária que tem valor 1 se a tarefa *i* foi programada como a *j*-ésima tarefa, e valor 0 caso contrário.

$S_{r,j}$ Variável que indica o instante em que a *j*-ésima tarefa começa a ser processada pela máquina *r*.

$C_{r,j}$ Variável que indica o instante em que a *j*-ésima tarefa termina de ser processada pela máquina *r*

T_j Variável que indica qual foi o atraso da *j*-ésima tarefa.

$Z_{r,j}$ Variável binária que indica se a estratégia de ligar e desligar foi implementada entre a tarefa na posição j e $j+1$ na máquina r

$U_{r,j}$, $W_{r,j}$ Variáveis intermediárias contínuas para a linearização da função objetivo não linear

4.2. Parâmetros

$p_{r,i}$ Tempo de processamento da tarefa *i* na máquina *r* (unidade de tempo).

d_i Prazo de entrega para a tarefa *i* (unidade de tempo).

s_r Instante em que começa o período de indisponibilidade na máquina *r* (unidade de tempo).

$P_{r,i}$ A potência de processamento da tarefa *i* na máquina *r* (unidade de energia).

P_r^{idle} A potência ociosa da máquina *r* (unidade de energia).

P_0 Potência comum, consumida pelos equipamentos e instalações auxiliares (unidade de energia).

N_r Número de vezes máximas de estratégias on/off na máquina *r*.

TB_r Tempo de *break-even* da máquina *r* (unidade de tempo).

$EnergyS_r$ Energia gasta para realizar uma operação de desligar e ligar a máquina *r* (unidade de energia).

M Valor suficientemente grande.

4.3. Função objetivo

Como visto anteriormente o objetivo de otimização principal será o consumo de energia, e portanto para computar este consumo podemos usar a seguinte expressão.

$$TEC' = \sum_{r=1}^{m} \sum_{j=1}^{n+m-1} \left(\left(1 - Z_{r,j}\right)\left(S_{r,j+1} - C_{r,j}\right)P_r^{idle} + EnergyS_r Z_{r,j} \right) + P_0\left(C_{m,n+m} - S_{1,1}\right) \quad (1)$$

Na expressão (1) temos o consumo total de energia (TEC), a primeira parte é a energia *IDLE* consumida, que contempla o tempo ocioso entre processamento de tarefas ou a energia gasta para desligar e ligar a máquina, se a estratégia for utilizada. A segunda parte é o gasto comum de energia, que corresponde a energia gasta para manter o ambiente com instalações auxiliares, que é calculada do momento em que a primeira tarefa começa a ser processada até o momento em que a última tarefa termina de ser processada.

Porém esta equação que vai ser utilizada na função objetivo é não-linear, pois contém o termo $(1 - Z_{r,j})(S_{r,j+1} - C_{r,j})P_r^{idle}$. Como isso dificultaria a otimização deste problema se faz necessário linearizar a equação, adicionando variáveis intermediárias $U_{r,j+1}$ e $W_{r,j}$ que substituiriam $(1 - Z_{r,j})S_{r,j+1}$ e $(1 - Z_{rj})C_{r,j}$ respectivamente. Desta forma o objetivo desta duas variáveis é apenas linearizar a equação (1), resultando na seguinte equação linearizada.

$$TEC = \sum_{r=1}^{m} \sum_{j=1}^{n+m-1} \left(\left(U_{r,j+1} - W_{r,j}\right)P_r^{idle} + EnergyS_r Z_{r,j} \right) + P_0\left(C_{m,n+m} - S_{1,1}\right) \quad (2)$$

Com isso podemos utilizar agora a esta equação (2) do consumo total de energia na função objetivo do modelo o que nos resulta na seguinte função objetivo.

$$min \quad TEC + \frac{\sum_{j=1}^{n+m} Tj}{UB_{Tj}(n+m)} \tag{3}$$

Onde TEC é a função linearizada do consumo de energia, equação (2), e o segundo termo presente na função objetivo diz respeito ao atraso total das tarefas, otimizado por uma abordagem lexicográfica, a ideia é por meio da utilização de um upper bound para o atraso, transformar o valor referente ao atraso total em um número entre 0 e 1 fazendo com que o modelo priorize a otimização do consumo de energia e depois ao atraso das tarefas, o upper bound foi calculado considerando o pior cenário possível, onde apenas uma operação é realizada por vez e o atraso é calculado com base na primeira tarefa a ser entregue.

$$UB_{Tj}(n+m) = \left(\sum_{r=1}^{m} \sum_{i=1}^{n+m} p_{r,i} - min_i\, d_i \right)(n+m) \tag{4}$$

4.4. Restrições

$$\sum_{j=1}^{n+m} x_{i,j} = 1 \qquad i = 1, 2, \ldots, n+m \tag{5}$$

$$\sum_{i=1}^{n+m} x_{i,j} = 1 \qquad j = 1, 2, \ldots, n+m \tag{6}$$

$$S_{r,j} - s_r \leq M\left(1 - x_{n+r,j}\right) \qquad r = 1, 2, \ldots, m\,;\; j = 1, 2, \ldots, n+m \tag{7}$$

$$S_{r,j} - s_r \geq -M\left(1 - x_{n+r,j}\right) \qquad r = 1, 2, \ldots, m\,;\; j = 1, 2, \ldots, n+m \tag{8}$$

$$S_{r,j+1} \geq S_{r,j} + \sum_{i=1}^{n+m} p_{r,i} x_{i,j} \qquad r = 1, 2, \ldots, m\,;\; j = 1, 2, \ldots, n+m-1 \tag{9}$$

$$S_{r+1,j} \geq S_{r,j} + \sum_{i=1}^{n+m} p_{r,i} x_{i,j} \qquad r = 1,2,\ldots,m-1;\ j = 1,2,\ldots,n+m \qquad (10)$$

$$S_{1,1} \geq 0 \qquad (11)$$

$$C_{r,j} = S_{r,j} + \sum_{i=1}^{n+m} p_{r,i} x_{i,j} \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m \qquad (12)$$

$$T_j \geq S_{m,j} + \sum_{i=1}^{n} x_{i,j} \left(p_{m,i} - d_i\right) \qquad j = 1,2,\ldots,n+m \qquad (13)$$

$$T_j \geq 0 \qquad j = 1,2,\ldots,n+m \qquad (14)$$

$$S_{r,j+1} - C_{r,j} \geq TB_r - M\left(1 - Z_{r,j}\right) \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m-1 \qquad (15)$$

$$S_{r,j+1} - C_{r,j} \leq TB_r + MZ_{r,j} \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m-1 \qquad (16)$$

$$\sum_{j=1}^{n+m-1} Z_{r,j} \leq N_r \qquad r = 1,2,\ldots,m \qquad (17)$$

$$U_{r,j+1} \geq S_{r,j+1} - MZ_{r,j} \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m-1 \qquad (18)$$

$$U_{r,j+1} \leq S_{r,j+1} + MZ_{r,j} \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m-1 \qquad (19)$$

$$U_{r,j+1} \leq M\left(1 - Z_{r,j}\right) \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m-1 \qquad (20)$$

$$U_{r,j} \geq 0 \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m \qquad (21)$$

$$W_{r,j} \geq C_{r,j} - MZ_{r,j} \qquad r = 1,2,\ldots,m;\ j = 1,2,\ldots,n+m-1 \qquad (22)$$

$$W_{r,j} \leq C_{r,j} + MZ_{r,j} \qquad r = 1,2,\ldots,m\,;\; j = 1,2,\ldots,n + m - 1 \qquad (23)$$

$$W_{r,j} \leq M\left(1 - Z_{r,j}\right) \qquad r = 1,2,\ldots,m\,;\; j = 1,2,\ldots,n + m - 1 \qquad (24)$$

$$W_{r,j} \geq 0 \qquad r = 1,2,\ldots,m\,;\; j = 1,2,\ldots,n + m \qquad (25)$$

$$X_{ij} \in \{0,1\} \qquad i,j = 1,2,\ldots,n + m \qquad (26)$$

O conjunto de restrições (5) vai garantir que cada tarefa esteja atribuída a somente uma posição, já o conjunto de restrições (6), por sua vez, irá garantir que cada posição esteja atribuída a uma única tarefa. Já o conjunto de restrições (7) e (8) vão garantir que o período de indisponibilidade da máquina *r* comece exatamente em s_r . O conjunto de restrições (9) garantem que, numa mesma máquina, a tarefa na posição *j* comece apenas quando a tarefa na posição anterior já tenha sido processada. O conjunto de restrições (10) vai garantir que uma tarefa apenas possa começar a ser processada em uma dada máquina se já foi processada na máquina anterior. A restrição (11) garante que a primeira tarefa comece somente após o instante 0.

O conjunto de restrições (12) serve para definir a variável que indica o instante de término das tarefas nas máquinas. O conjunto de restrições (13) vai garantir que o atraso seja maior ou igual à diferença do tempo de conclusão e a entrega de um determinada tarefa, já o conjunto de restrições (14) garante que o atraso de uma tarefa seja maior ou igual a zero, não assumindo valores negativos.

O conjunto de restrições (15) e (16) vão restringir a estratégia de ligar e desligar a máquina, ou seja, quando o tempo *IDLE* ($S_{r,j+1} - C_{r,j}$) é maior que o chamado "ponto de equilíbrio", TB_r, ela pode ser desligada, caso o contrário não. O conjunto de restrições (17) garante que não haja mais do que o número máximo permitido de utilização da estratégia de ligar e desligar a máquina. Os conjuntos de restrições (18),(19),(20),(21) vão garantir que $U_{r,j+1} = (1 - Z_{r,j})S_{r,j+1}$ seja sempre verdade, isto é, se $Z_{r,j}$= 0 então os conjuntos (18) e (19) vão garantir que $U_{r,j+1}$ seja maior ou igual e menor ou igual à $S_{r,j+1}$ ao mesmo tempo,

desta forma assumindo seu valor. Agora se $Z_{r,j}$= 1 os conjuntos (20) e (21) vão garantir que $U_{r,j+1}$ seja maior ou igual e menor ou igual à 0, desta forma assumindo o valor 0. Semelhantemente as restrições (22),(23),(24),(25) vão garantir que $W_{r,j} = (1 - Z_{r,j}) C_{r,j}$ seja sempre verdade, isto é se $Z_{r,j}$= 0 então $W_{r,j}$ assume o valor de $C_{r,j}$. Se $Z_{r,j}$= 1, então $W_{r,j}$ assume o valor 0.

4.5. Instância exemplo

Para avaliarmos o modelo descrito anteriormente, podemos utilizar uma instância de teste e encontrar a solução ótima deste problema, isto é, minimizar a energia consumida ao processarmos a tarefa e o atraso total das mesmas. Para este exemplo consideramos um ambiente com 3 máquinas que deve processar 6 tarefas os tempos de processamento de cada uma dessas tarefa nas 3 máquinas é dado a seguir:

Tabela 1: Instância exemplo com 6 tarefas e 3 máquinas

J_i	p_{1i}	p_{2i}	p_{3i}	d_i
J_1	2	1	5	8
J_2	3	3	3	10
J_3	3	5	4	14
J_4	4	3	2	17
J_5	2	2	4	21
J_6	3	4	6	29

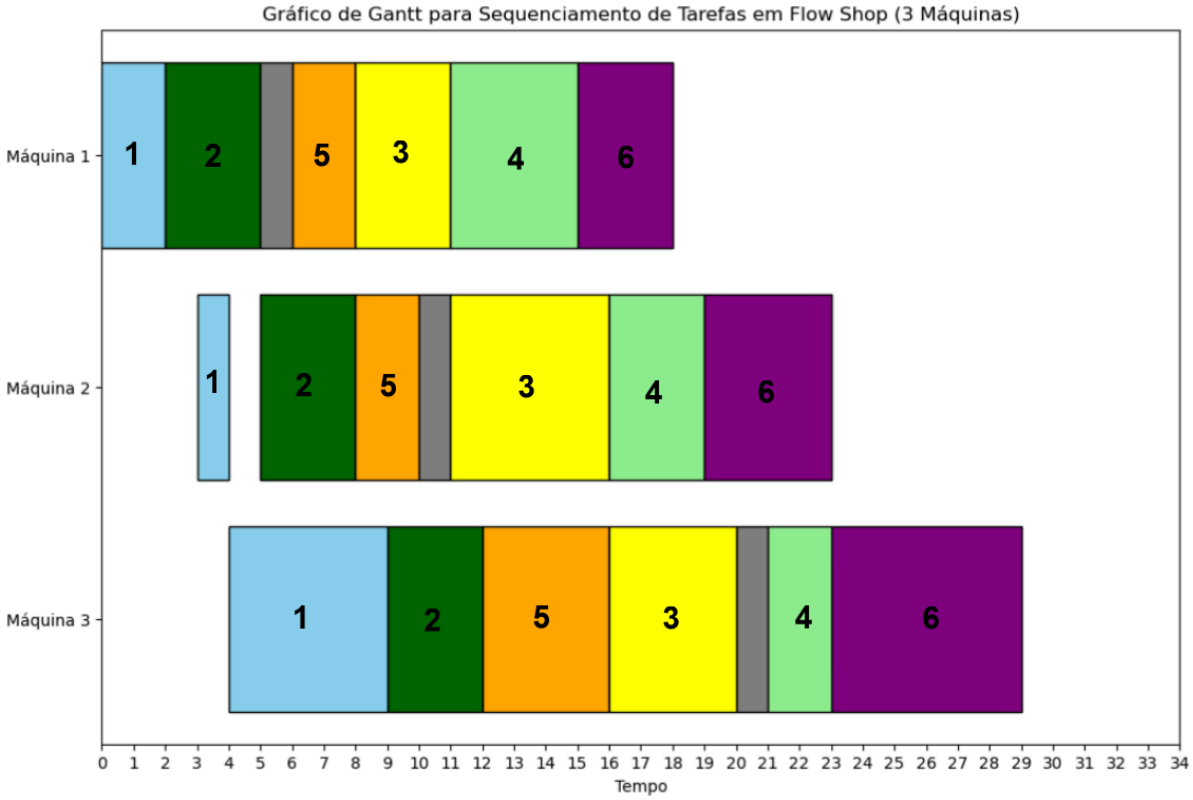
Fonte: elaborado pelo autor

Neste exemplo iremos considerar que cada máquina possui um período de indisponibilidade, na máquina 1 o período de indisponibilidade começa em 5 (s_1 = 5) e tem duração de uma unidade de tempo, o período de indisponibilidade da máquina 2 começa em 10 (s_2 = 10) e tem duração de uma unidade de tempo, por fim, na máquina 3 no período de indisponibilidade começa em 20 (s_3 = 20) e também tem duração de 1 unidade de tempo.

Além dos parâmetros já citados, aqui iremos considerar que a energia consumida em tempo *IDLE* pelas máquinas é de 2 unidades de energia, o consumo quando se utiliza a estratégia de ligar e desligar as máquinas é 1 unidade de energia. E o custo comum, isto é, para os equipamentos auxiliares é de 1 unidade de energia.

Aplicando no modelo matemático chegamos a uma solução ótima que tem como sequenciamento tarefa 1, tarefa 2, tarefa 5, tarefa 3, tarefa 4, tarefa 6. O valor da função objetivo encontrado foi de 30,03, onde a parte inteira representa a energia consumida e os decimais representam a otimização do atraso total, neste exemplo obtemos um consumo de 30 unidades de energia e o atraso total obtido de 15 unidades de tempo. Podemos perceber pelo sequenciamento presente na Figura 3, que a estratégia de ligar e desligar a máquina foi utilizada entre a tarefa 1 e tarefa 2 na máquina 2.

Figura 3: Gráfico Gantt do sequenciamento ótimo da instância exemplo minimizando o gasto de energia e atraso total

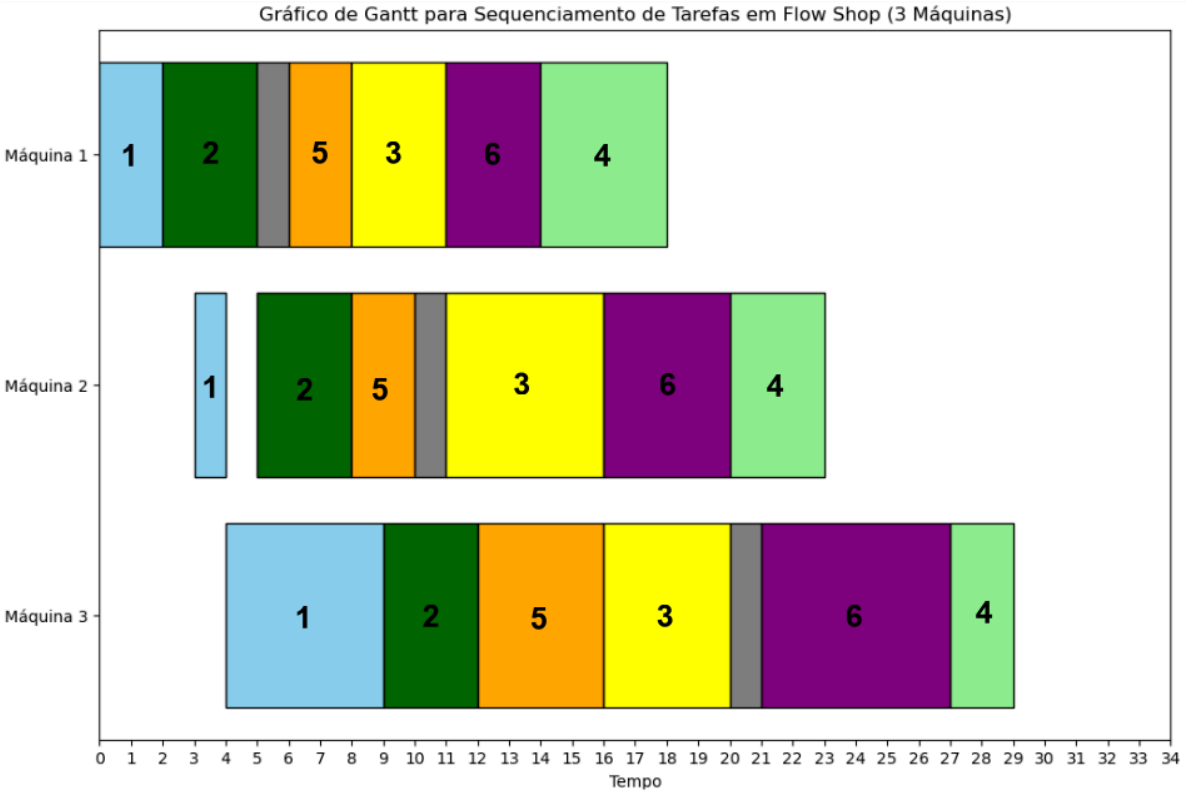


Fonte: Matplotlib Python

Com base nesse exemplo podemos também evidenciar o funcionamento da otimização lexicográfica, que tem como foco a minimização da energia total gasta, mas também mantém em vista o atraso total. Podemos destacar seu funcionamento retirando da função objetivo a expressão responsável por calcular o atraso total, otimizando o problema

apenas na ótica de energia. A solução ótima obtida então tem o seguinte sequenciamento: tarefa 1, tarefa 2, tarefa 5 , tarefa 3, tarefa 6, tarefa 4. Com o valor da função objetivo de 30, logo, temos o mesmo consumo energético quando comparado a primeira solução, porém o atraso total obtido na solução da Figura 4 é de 21, tendo um aumento de 6 unidades quando comparado a primeira solução.

Figura 4: Gráfico Gantt do sequenciamento ótimo da instância exemplo minimizando o gasto de energia



Fonte: Matplotlib Python

5. HEURÍSTICA

Diante da crescente complexidade dos problemas de programação da produção, como é o caso dos ambientes *flow shop* com restrições de disponibilidade, muitas vezes torna-se inviável encontrar soluções exatas em um tempo computacional razoável. Isso ocorre porque esses problemas se enquadram na classe *NP-hard,* ou seja, à medida que o tamanho do problema aumenta, o esforço necessário para encontrar a solução ótima cresce exponencialmente. Diante dessa limitação, torna-se essencial o uso de heurísticas como abordagem prática para tomada de decisão. A área de produção é uma das áreas que mais utiliza métodos heurísticos (Zanakis *et al*. 1989).

Heurísticas são procedimentos para resolver problemas através de um enfoque "intuitivo", em geral racional, no qual a estrutura do problema possa ser interpretada explorada inteligentemente para obter uma solução razoável (Nicholson, 1971). Portanto, são métodos ou estratégias simplificadas que não garantem a solução ótima, mas conseguem encontrar soluções satisfatórias e próximas do ideal em tempo viável. Em vez de explorar exaustivamente todas as combinações possíveis, esses métodos procuram identificar padrões e atalhos que conduzam a soluções de alta qualidade, mantendo a viabilidade operacional.

5.1. Heurística construtiva

Para a construção de uma solução inicial iremos utilizar uma heurística construtiva que irá levar em conta aspectos importantes da formulação do problema para montar um sequenciamento de tarefas. Heurísticas construtivas constroem uma solução de forma incremental, adicionando elementos passo a passo até formar uma solução completa. Essas heurísticas seguem uma abordagem sequencial, onde cada decisão feita em uma etapa impacta as etapas seguintes.

A heurística que será utilizada está baseada na proposta por Ronconi & Henriques (2009), que faz o uso da regra de despacho FPD (*fitting processing times and due dates*). Utilizaremos uma regra de despacho com um funcionamento dinâmico, ou seja, depois da seleção de uma dada tarefa para uma determinada posição no sequenciamento, uma lista de prioridade é calculada considerando todas tarefas que ainda não foram posicionadas. A escolha de uma tarefa está diretamente ligada com a tarefa que foi posicionada previamente, desta forma para iniciarmos o algoritmo precisamos selecionar a primeira tarefa para que a lista de prioridade seja calculada, uma abordagem que pode ser utilizada para a escolha dessa

tarefa inicial é a tarefa que tenha a menor soma da data de entrega e tempo de processamento na primeira máquina (I_k), fazendo isso estamos levando em consideração uma das características do problema que é a minimização do atraso total.

$$I_k = d_k + p_{1,k}$$

Para as tarefas seguintes, iremos utilizar uma medida de prioridade (F_k), onde a tarefa com menor valor dessa medida será escolhida como a próxima a ser posicionada. Essa medida de prioridade considera aspectos relacionados ao tempo de processamento e data de entrega das tarefas candidatas, ela é calculada por meio de dois termos, o primeiro termo (aproveitamento de janela) vai beneficiar tarefas que possuam um melhor "encaixe" com as janelas deixadas pela última tarefa posicionada em cada máquina, visto que o objetivo de otimização principal é o de minimização de consumo energético é de extrema importância que as tarefas sejam alocadas de forma a reduzir o tempo ocioso das máquinas. Já o segundo termo (folga dinâmica) visa aumentar a prioridade de tarefas que estejam próximas de sua data de entrega, trazendo um olhar para o atraso total.

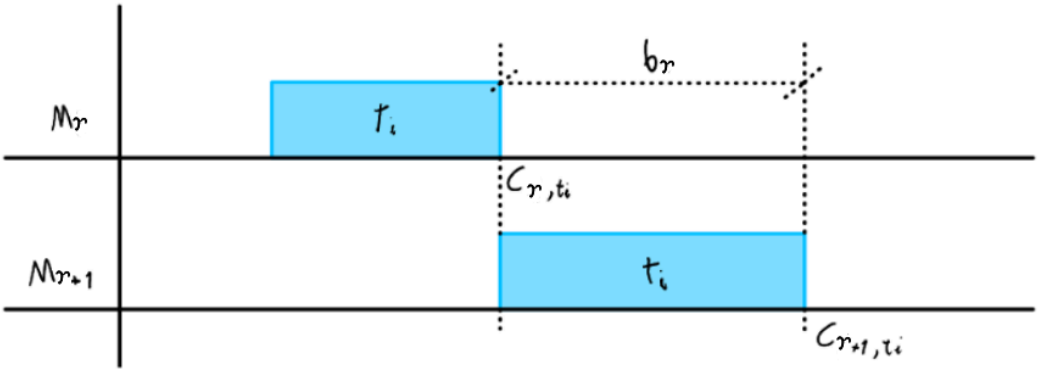
5.1.1. Aproveitamento de janela (*fit*)

Para calcular o aproveitamento da janela que uma tarefa candidata *k* terá em relação a tarefa anteriormente posicionada precisamos levar em conta a janela em uma dada máquina (b_r) assim como o tempo de processamento da tarefa a ser posicionada (p_{rk}), queremos a tarefa que melhor se encaixe nessas janelas, portanto podemos medir esse encaixe a partir da soma da diferença absoluta entre a janela e o tempo de processamento da tarefa a ser posicionada em cada máquina.

$$fit = \sum_{r=1}^{m-1} \left| b_r - p_{rk} \right|$$

A janela de disponibilidade de cada máquina (b_r), diz respeito ao espaço de tempo deixado para realizarmos uma tarefa quando outra já está posicionada, a maneira mais intuitiva de pensarmos nessa janela é a partir do instante de tempo que a tarefa já posicionada (t_i) acaba de ser processada em uma máquina *r+1* ($C_{r+1,ti}$) e subtraí-lo do instante tempo que esta mesma tarefa termina na máquina anterior *r* ($C_{r,ti}$). Na Figura 5, temos a representação dessa janela de tempo.

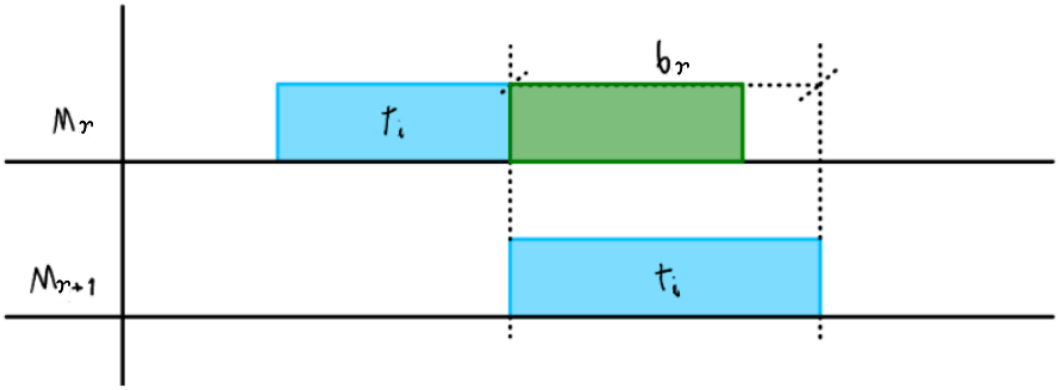
Figura 5: Janela de disponibilidade



Fonte: elaborado pelo autor

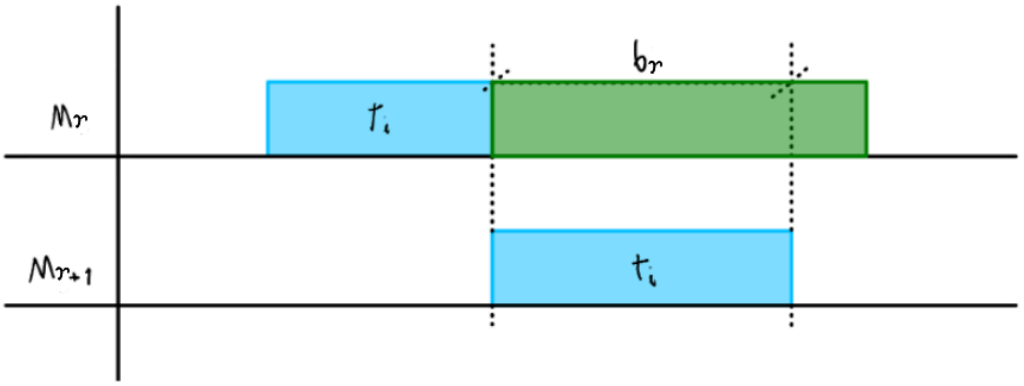
A ideia é a partir disso escolher a tarefa que vai melhor se encaixar nesse espaço deixado pela tarefa t_i de modo a evitar possíveis tempos ociosos como descritos na Figura 6 e Figura 7.

Figura 6: Tarefa menor do que janela disponível



Fonte: elaborado pelo autor

Figura 7: Tarefa maior do que janela disponível

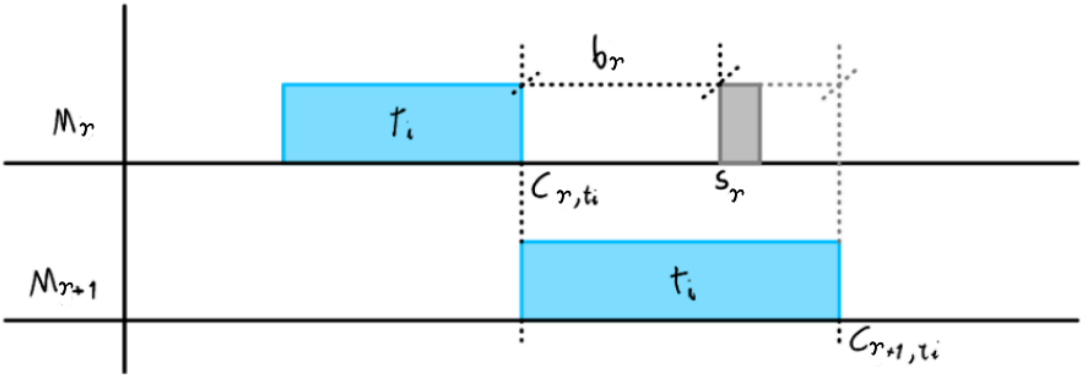


Fonte: elaborado pelo autor

Considerando o problema tratado neste trabalho, precisamos também levar em conta que as máquinas terão tempos de indisponibilidade que irão interferir diretamente nestas

janelas, visto que estamos tratando do caso *non-resumable* e assim uma vez que as tarefas são iniciadas não podem ser interrompidas no meio de seu processamento. Portanto é de extrema importância ter em vista o cenário descrito pela Figura 8, onde a indisponibilidade que se inicia em (s_r) vai fazer com que a janela da máquina *r* seja menor do que se inicialmente pensava, o seu cálculo então será a diferença entre o instante que a indisponibilidade se inicia (s_r) subtraído do instante em que a tarefa já posicionada termina na máquina *r* ($C_{r,ti}$) .

Figura 8: Janela disponível considerando indisponibilidade de máquinas

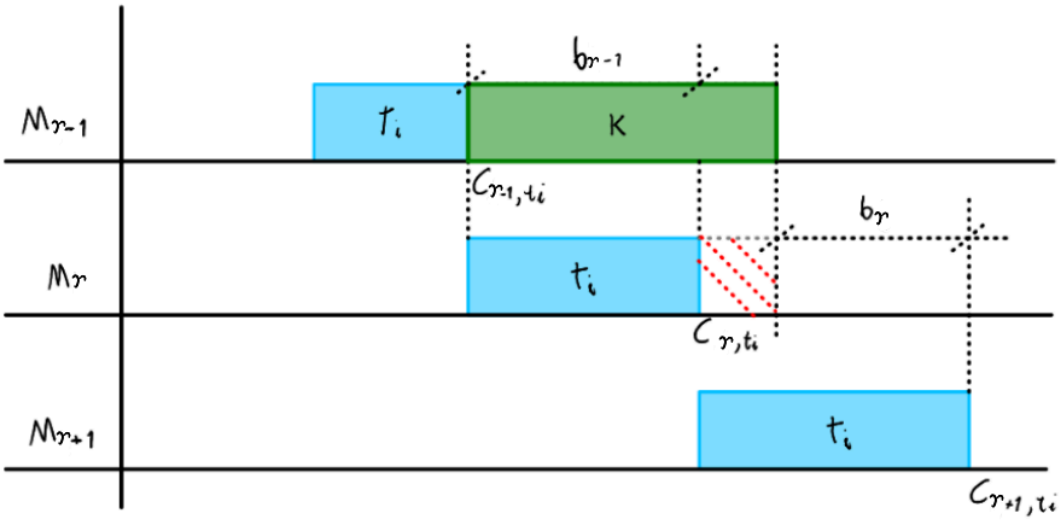


Fonte: elaborado pelo autor

Outro ponto importante é que, com exceção da primeira máquina, o modo de se calcular a janela (b_r) pode variar de acordo com o encaixe da tarefa *k* na máquina anterior, isto é, se tivermos a situação observada na Figura 6 o cálculo deve ser feito da maneira como foi já foi descrito anteriormente, porém se tivermos a situação que é observada na Figura 7, precisamos levar em conta que a tarefa *k* só estará disponível para ser realizada na máquina *r*+1 quando ela terminar de ser processada na máquina *r,* então se considerarmos a janela de disponibilidade começando assim que a tarefa já posicionada (t_i) termina na máquina *r* estaremos atribuindo a esta janela um espaço que não poderá ser utilizado.

Desta forma como podemos observar na Figura 9, para a máquina *r*-1 o cálculo da janela será feito normalmente, porém quando analisarmos a janela da máquina *r* devemos subtrair do instante de término da tarefa t_i na máquina *r*+1 o instante em que a tarefa *k* termina de ser processada na máquina *r*-1, e assim a janela b_r não irá considerar o espaço que não pode ser utilizado, hachurado em vermelho. O instante em que a tarefa *k* termina numa máquina pode ser considerado como o instante que a tarefa t_i termina na primeira máquina somado dos tempos de processamento da tarefa k nas máquinas anteriores a aquela que se calcula a janela.

Figura 9: Ajuste da janela disponível considerando espaço inutilizado



Fonte: elaborado pelo autor

A partir de todas características das janelas que foram descritas, podemos chegar numa expressão para o cálculo da mesma, para a primeira máquina teremos:

$$b_1 = min_{>0}\left\{ C_{2,t_i} - C_{1,t_i},\ s_1 - C_{1,t_i} \right\}$$

Para as máquinas seguintes:

$$b_j = min_{>0}\left\{ C_{r+1,ti} - max\left\{ C_{r,ti}, C_{1,k} + \sum_{w=1}^{r-1} p_{w,k} \right\},\ s_r - max\left\{ C_{r,ti}, C_{1,k} + \sum_{w=1}^{r-1} p_{w,k} \right\} \right\}$$

5.1.2. Folga dinâmica

O segundo termo utilizado para compor a medida de prioridade é o referente a folga dinâmica, este termo é importante para que tarefas que estejam mais próximas a sua data de entrega tenham a sua prioridade aumentada, isso vai de encontro com o outro objetivo de otimização do problema deste trabalho que é a minimização do tempo total de atraso. O cálculo deste termo será dado por:

$$dynslack = \left(LB_k - C_{1,ti}\right)$$

Onde LB_k representa um limite inferior para a entrega da tarefa k, que pode ser calculado como a diferença entre o instante em que a tarefa deve ser entregue e a soma dos tempos de processamento da mesma em todas as máquinas.

$$LB_k = d_k - \sum_{r=1}^{m} p_{r,k}$$

Com este segundo termo estabelecido podemos então fazer a definição do cálculo da medida de prioridade, juntando o *fit* com *dynslack* teremos a medida de prioridade que pode ser ajustada quando ao peso dado para cada termo por meio de um parâmetro ρ.

$$F_k = \rho\left(\sum_{r=1}^{m-1} \left|b_r - p_{rk}\right|\right) + (1 - \rho)\left(LB_k - C_{1,ti}\right)$$

5.1.3. Pós-processamento

Depois que todas as tarefas já tiverem sidos posicionadas de acordo com a lista de prioridade calculada dinamicamente, iremos obter o sequenciamento das tarefas, porém para melhor adequarmos a solução ao problema estudado, será aplicado um pós-processamento que vai ajustar o instante em que cada tarefa deve começar, respeitando a ordem que foi obtida. Este passo tem como objetivo, agrupar as tarefas para que o tempo ocioso das máquinas seja o menor possível, e consequentemente a energia total gasta também.

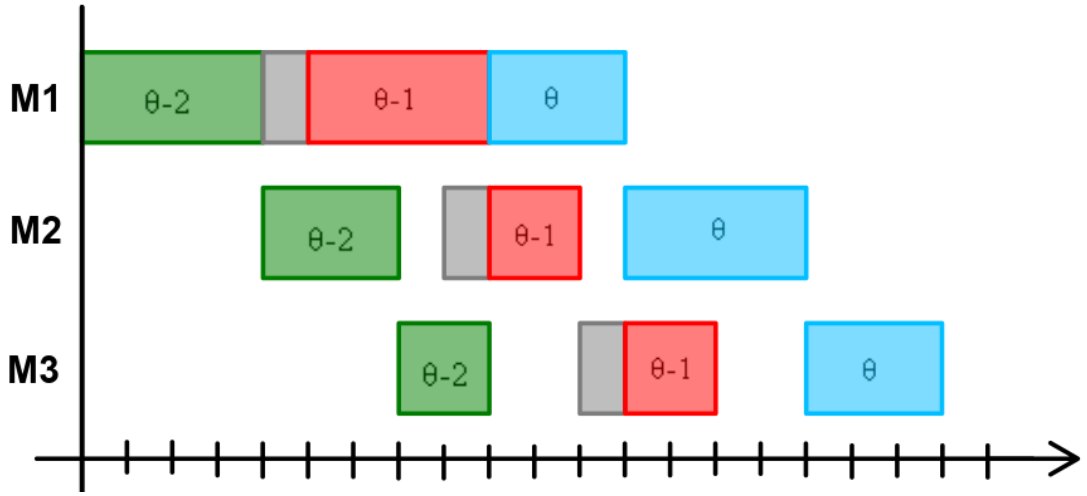
Começando pela última máquina, as tarefas serão ajustadas de acordo com o instante de início da última tarefa posicionada (tarefa na posição θ). Logo o instante de término da tarefa na posição θ-1 será ajustado para o instante em que a tarefa na posição θ inicia, e o instante de início recalculado com base no seu tempo de processamento, o mesmo será feito para a tarefa θ-2, porém agora com base no instante de início da tarefa na posição θ-1, e assim por diante, até que todas as tarefas sejam ajustadas, vale lembrar que em cada máquina iremos possuir períodos de indisponibilidade, desta forma as tarefas devem ser ajustadas respeitando essas janelas.

Nas máquinas seguintes isso deve ser feito seguindo a mesma ideia que foi aplicada na última máquina, porém o ajuste da tarefa na posição θ-1 numa máquina *r* não deve considerar apenas o instante de início da tarefa uma posição acima (θ) mas também o instante de início da tarefa na posição θ-1 na máquina *r*+1, visto que por estarmos considerando um ambiente *flow shop* uma tarefa só pode iniciar seu processamento na máquina *r*+1 se já foi completada na máquina *r*. Portanto o instante de término da tarefa na posição θ-1 deve assumir o menor valor entre o instante de início da tarefa na posição θ na

máquina *r* e instante de início da tarefa na posição θ-1 na máquina *r*+1, sempre respeitando as janelas de indisponibilidade.

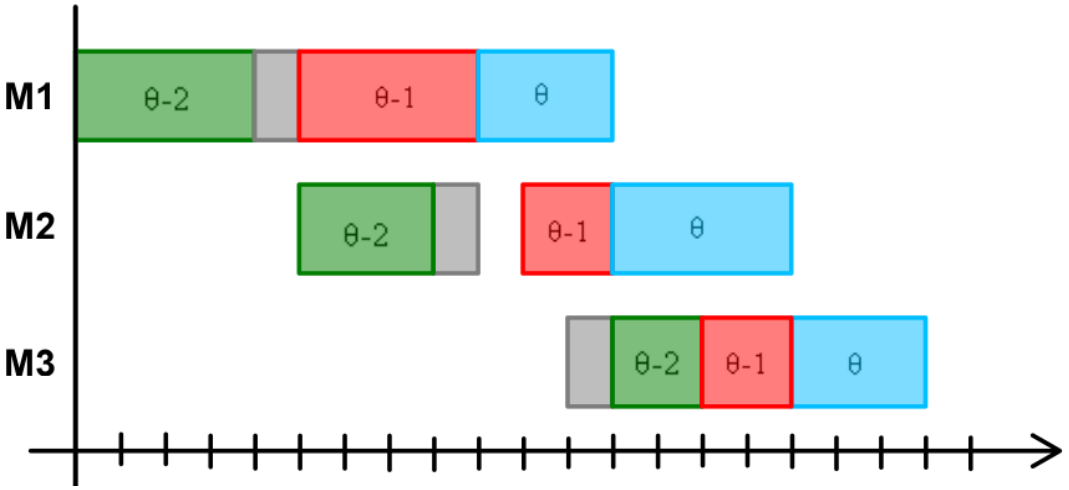
Após o pós processamento teremos o sequenciamento final que será utilizado para calcular o atraso total e consumo total de energia, aplicando a estratégia liga e desliga removendo os maiores tempos ociosos entre tarefas, desde que respeitem o tempo de *break-even* e número máximo de utilização da estratégia liga e desliga por máquina. Na Figura 10 temos um exemplo de um sequenciamento de três tarefas em três máquinas obtido posicionando as tarefas assim que as mesmas possam ser processadas na máquina seguinte e na Figura 11 temos outro sequenciamento que segue a mesma ordem de tarefas, porém nesse caso, o pós-processamento foi aplicado e as tarefas estão dispostas de modo a minimizar os tempos ociosos das máquinas.

Figura 10: Exemplo de sequenciamento antes do pós-processamento



Fonte: elaborado pelo autor

Figura 11: Exemplo de sequenciamento depois do pós-processamento



Fonte: elaborado pelo autor

5.2. Heurística de melhoria

A fim de melhorar a solução obtida pela heurística construtiva, iremos utilizar uma heurística de melhoria baseada em busca local. Heurísticas de melhoria são técnicas utilizadas para refinar uma solução inicial existente, melhorando-a por meio de pequenas alterações controladas, essas heurísticas operam principalmente com métodos de busca local. A busca local é uma abordagem amplamente usada para resolver problemas de otimização difíceis. Um problema de otimização tem um conjunto de soluções e uma função de custo que atribui um valor numérico a cada solução. O objetivo é encontrar uma solução ótima, uma que tenha o custo mínimo, ou máximo, a depender do problema (Aarts & Lenstra, 2003).

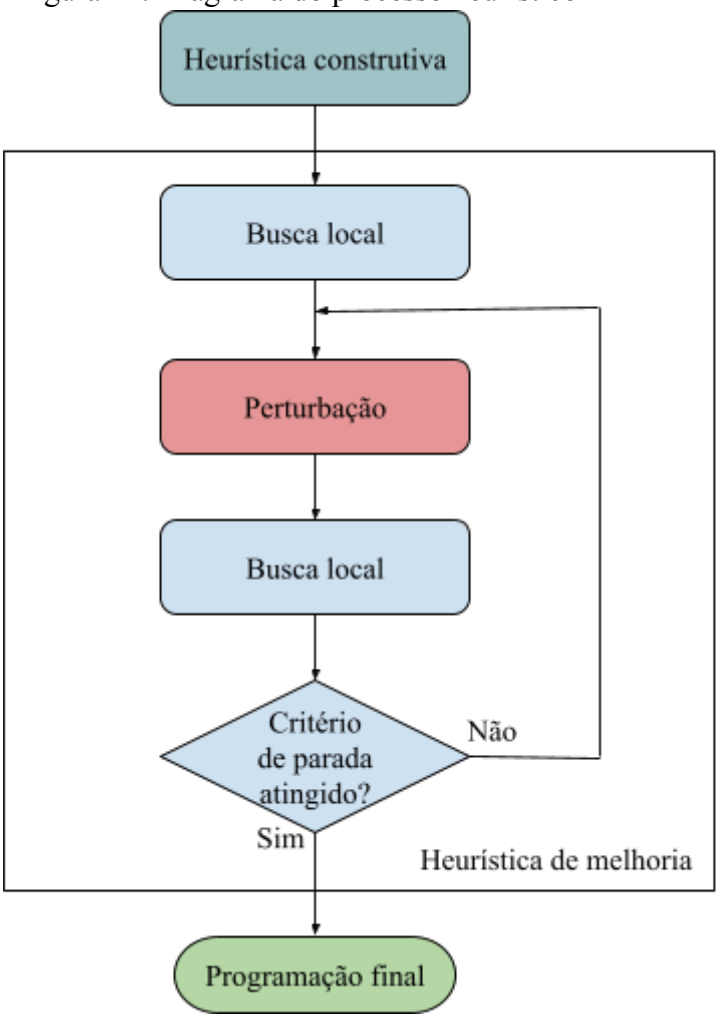
Desta forma utilizaremos a heurística construtiva como ponto de partida e com base nela construir uma vizinhança de soluções. O movimento escolhido para gerar as vizinhanças será o de inserção, retirando um elemento da sua posição e o inserindo nas demais, por exemplo, vamos considerar uma solução *x* dada por uma permutação de tarefas de 1 a 4. Então se x = (2,3,1,4) a vizinhança *N(x)* gerada pelo movimento de inserção será *N(x)* = {(3,**2**,1,4), (3,1,**2**,4), (3,1,4,**2**), (2,1,**3**,4), (2,1,4,**3**), (**1**,2,3,4), (2,3,4,**1**), (**4**,2,3,1), (2,**4**,3,1)}.

Após gerada a vizinhança, escolhemos a sequência que irá apresentar a menor energia total gasta, caracterizando um movimento *best-move*, onde se avalia todas as possíveis vizinhanças da solução atual e se escolhe a que oferece o maior ganho antes de efetuar o movimento. Com essa nova sequência escolhida iremos repetir o processo de gerar a vizinhança até que não haja nenhuma solução melhor do que a sequência geradora da vizinhança, atingindo o critério de parada da busca local.

Por fim, foi adicionado uma perturbação juntamente com a repetição da etapa de busca local, com intuito de fugir de ótimos locais. Em buscas locais temos uma forte influência da sequência de partida (gerada pela heurística construtiva), essa perturbação pode nos permitir alcançar uma melhor solução ao aplicar uma mudança aleatória na sequência. Para isso foi utilizado o movimento de troca aleatória entre elementos da sequência alcançada na busca local anterior. Com *n* referindo-se ao número de tarefas, são realizadas $\frac{n}{2}$ trocas, sendo esse resultado sempre arredondado para cima, então por exemplo, se tivermos 5 tarefas, 3 trocas serão feitas, se tivermos 9 tarefas, 5 trocas serão feitas e assim por diante. Após realizada a troca o processo de busca local é feito novamente e como saída teremos a solução incumbente. Este processo de perturbação e busca local é refeito até que a solução

incumbente passe a ter uma melhora menor do que 0.5%, assim atingido o critério de parada. Na figura 12 temos um diagrama que representa o fluxo do processo heurístico empregado.

Figura 12: Diagrama do processo heurístico



Fonte: elaborado pelo autor

6. EXPERIMENTOS NUMÉRICOS

Foram realizados testes, utilizando instâncias geradas, para analisar os resultados obtidos utilizando o modelo matemático, assim como os métodos heurísticos, a fim de comparar a qualidade das soluções obtidas e o tempo computacional necessário para cada uma das abordagens.

Todos os experimentos foram resolvidos em um computador Intel Core i7 3.40 GHz e com memória RAM de 16 GB, a linguagem de programação utilizada foi Python e o modelo matemático foi implementado utilizando a ferramenta Gurobi 11.0.3.

6.1. Instâncias

Para realizarmos os testes computacionais são necessários dados que representem o ambiente de produção em que estamos modelando os métodos de solução, e para tal utilizaremos instâncias de diferentes tamanhos baseadas em trabalhos retirados da literatura que irão contemplar diferentes cenários para uma posterior análise.

Devido a complexidade do problema aqui tratado, como queremos fazer a utilização de um modelo matemático, ou seja um método exato, ao aumentarmos a quantidade de tarefas e máquinas presente no problema o tempo computacional é cada vez mais requerido. Desta forma foram geradas diferentes instâncias que se adequam ao tamanho do problema. As instâncias geradas para teste e validação dos métodos foram baseadas em Mouzon *et al*. (2019), com tamanhos de 5, 6, 7, 8 tarefas em ambientes de 5, 6 e 7 máquinas e tamanhos de 7, 8 e 9 tarefas para ambiente de 8 máquinas, desta forma 15 diferentes tamanhos de problema. Para cada tamanho foram geradas 3 instâncias, totalizando assim 45 experimentos. Já para os testes de maior escala, onde os métodos heurísticos serão testados em situações que podem também ser encontradas na prática, as instâncias foram retiradas de Taillard (1993) com tamanhos de problemas com 20, 50, 100 e 200 tarefas para um ambiente de 10 máquinas.

Os tempos de processamento das tarefas foram gerados a partir de uma distribuição discreta uniforme entre 1 e 99. O tempo de indisponibilidade de cada máquina foi definido com base em Aggoune (2004), assumindo o valor da média dos tempos de processamento de cada máquina, desta forma teremos:

$$p_{r,\,n+r} = \frac{\sum_{i=1}^{n} p_{r,i}}{n}$$

Já o início do tempo de indisponibilidade de cada máquina foi definido por uma adaptação do que temos em Xu *et al*. (2018), generalizando o caso de duas máquinas para múltiplas máquinas. No caso de duas máquinas s_r é definido pela soma dos tempos de processamentos das tarefas na máquinas *r* dividido por dois, para o caso de múltiplas máquinas iremos adicionar a soma dos tempos de indisponibilidade das máquinas anteriores, portanto teremos que:

$$s_r = \frac{\sum_{i=1}^{n} p_{r,i}}{2} + \sum_{k=1}^{r} p_{k,n+k}$$

As datas de entregas serão geradas seguindo Armentano & Ronconi (1999), onde essas datas são distribuídas uniformemente entre $P(1 - T - R/2)$ e $P(1 - T + R/2)$. Nestas equações T e R representam o fator de atraso das tarefas e a faixa de dispersão das datas de entrega, respectivamente, desta forma quando variamos os valores desses dois parâmetros temos diferentes cenários. *P* representa um *lower bound* para o *makespan*, partindo da definição feita por Taillard (1993) e adaptando para melhor adequarmos ao problema deste trabalho, isto é, levar em consideração a restrição de disponibilidade, podemos calcular *P* como:

$$P = max\{ max_{1\leq r\leq m}\{\sum_{i=1}^{n} p_{r,i} + p_{r,n+r} + min_i \sum_{k=1}^{r-1} p_{k,i} + min_i \sum_{k=r+1}^{m} p_{k,i}\},$$

$$max_i \sum_{r=1}^{m} p_{r,i}\}$$

Neste trabalho optou-se por utilizar um cenário mais crítico para as datas de entrega, com alto fator de atraso e pequena faixa de datas de entrega, desta forma ao gerarmos as instâncias foi considerado T = 0,4 e R = 0,6. Além disso, os demais parâmetros considerados foram baseados em Meng *et al.* (2019), a potência comum e a potência ociosa das máquinas são 1 e 2 respectivamente. O número máximo de utilização da estratégia liga e desliga por máquina é 3 e o tempo de *break-even* das máquinas é 20.

6.2. Resultados e discussão

Os resultados obtidos a partir da utilização das instâncias nos permitirão validar e avaliar os métodos explorados neste trabalho, primeiramente as instâncias serão resolvidas utilizando o método exato e os métodos heurísticos propostos, após isso será realizado uma análise de sensibilidade em relação ao tamanho das janelas de indisponibilidade das máquinas e por fim os métodos heurísticos serão aplicados as instâncias de maior escala para validação em cenários mais próximos da prática.

6.2.1. Método exato

A Tabela 1 apresenta os resultados obtidos utilizando o modelo matemático da seção 4 para os 45 experimentos, a otimização do modelo tinha um limite de 30 minutos de tempo de execução, desta forma para os experimentos que extrapolaram esse limite de tempo e não alcançaram a solução ótima, temos o *gap* que representa diferença entre o limite inferior e o limite superior da função objetivo. A média do *gap* obtido foi de 1,61%, sendo que 26 dos 45 experimentos tiveram *gap* igual a 0, ou seja, foi possível encontrar a melhor solução possível, e o maior *gap* obtido foi de 16,5 % para a segunda instância do problema com 8 tarefas e 7 máquinas.

Podemos perceber que ao aumentarmos o número de máquinas o tempo computacional aumenta substancialmente, por exemplo, considerando os problemas com 7 tarefas, para um ambiente de 5 máquinas a média do tempo computacional das 3 instâncias foi de aproximadamente 53 segundos, já para o ambiente de 6 máquinas essa média foi de 748 segundos. Essa tendência confirma a complexidade do problema e a dificuldade da utilização do modelo matemático quando escalamos o tamanho dos experimentos. Vale ressaltar também que os resultados obtidos para o atraso total não representam o valor ótimo, isto é, como foi utilizado a otimização lexicográfica primeiro otimizamos a energia total gasta, para que assim o atraso total seja otimizado de modo que a programação da produção tenha o valor que já havia sido obtido para o objetivo referente à energia consumida.

Tabela 2: Resultado computacionais utilizando o método exato

Tarefas	Máquinas	Instância	Método exato			
			Tempo computacional (segundos)	Energia (UE)	Atraso Total (UT)	GAP (%)
5	5	1	33	636	1089	0
5	5	2	8	657	1333	0
5	5	3	10	632	917	0
6	5	1	14	665	1135	0
6	5	2	23	657	828	0
6	5	3	90	550	866	0
7	5	1	25	587	895	0
7	5	2	99	759	1407	0
7	5	3	36	637	1435	0
8	5	1	77	810	1536	0
8	5	2	326	740	1119	0
8	5	3	275	609	1232	0
5	6	1	843	665	1015	0
5	6	2	232	743	884	0
5	6	3	647	738	1115	0
6	6	1	1800	818	1271	0,98
6	6	2	1800	856	1150	0,23
6	6	3	98	815	1721	0
7	6	1	601	838	1656	0
7	6	2	900	763	1578	0,51
7	6	3	744	868	1231	0
8	6	1	253	841	1653	0
8	6	2	899	743	1435	0
8	6	3	1106	746	2167	0
5	7	1	258	756	1026	0
5	7	2	745	725	1180	0
5	7	3	1254	617	551	0
6	7	1	222	841	2216	0

6	7	2	1800	889	1670	1
6	7	3	1800	732	1641	1,64
7	7	1	305	789	2542	0
7	7	2	1800	729	1771	0,74
7	7	3	1800	770	2098	1,41
8	7	1	1800	853	2724	0,58
8	7	2	1800	945	2040	16,5
8	7	3	1800	904	1774	6,6
7	8	1	1800	968	1276	0,87
7	8	2	1800	936	1976	0,52
7	8	3	1800	901	1389	6,21
8	8	1	1800	943	2253	3,6
8	8	2	1800	1009	2916	0,8
8	8	3	1800	1069	2138	6,8
9	8	1	1800	993	2687	9,3
9	8	2	1800	1011	2916	12,2
9	8	3	1800	1019	2367	2
Média						1,61

Fonte: elaborado pelo autor

6.2.2. Métodos heurísticos

Para executar os testes com a heurística construtiva, primeiramente foi realizado uma análise de sensibilidade do parâmetro ρ utilizado para ajustar os pesos do cálculo da medida de prioridade utilizado no método heurístico, desta forma variando ρ no intervalo de 0 e 1 com passos de 0,2 chegamos no *gap* médio da energia e o gap médio do atraso (%), fazendo a comparação entre os resultados obtidos para energia total gasta (FE) e resultados obtidos para atraso total (FA), utilizando a heurística (HC) e o modelo matemático (MM), calculados da seguinte forma.

$$GAP\ Energia(\%) = \frac{FE_{HC} - FE_{MM}}{FE_{MM}}$$

$$GAP\ Atraso(\%) = \frac{FA_{HC} - FA_{MM}}{FA_{MM}}$$

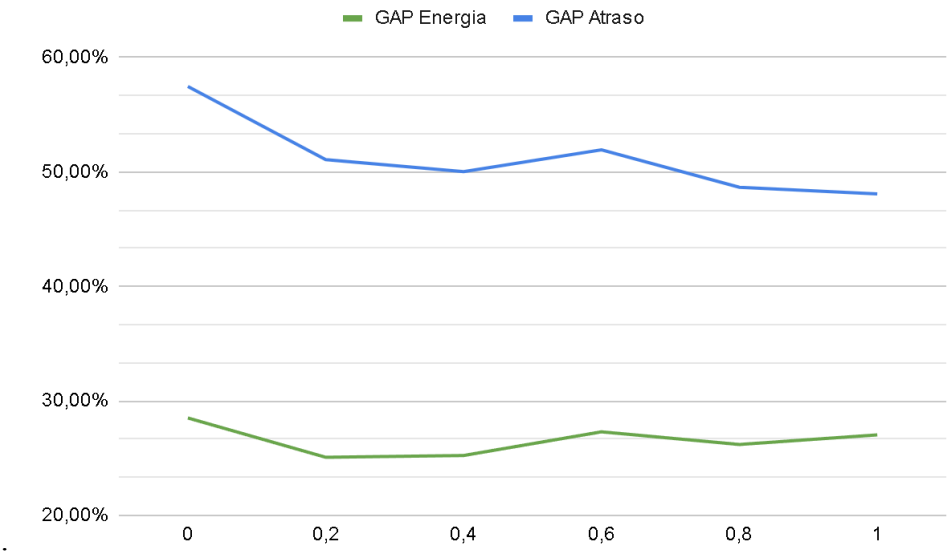
A Tabela 3 apresenta os resultados obtidos para cada valor de parâmetro utilizado, e a Figura 13 traz a representação gráfica. Analisando os 3 melhores resultados quanto ao *gap* de energia, principal critério a ser otimizado, temos os valores de 0,2, 0,4 e 0,8 com *gaps* da energia de 25,08%, 25,23% e 26,19% respectivamente. A diferença percentual entre esses valores está em aproximadamente 1%, o que demonstra a robustez da heurística quanto a diferentes ρ, e portanto para a posterior análise o valor do parâmetro que será utilizado é 0,5, distribuindo igualmente os pesos para as duas componentes que compõem a medida de prioridade da heurística construtiva.

Tabela 3: Relação entre a média dos gaps obtidos e o ρ utilizado na heurística construtiva

ρ	**GAP Energia (%)**	**GAP Atraso (%)**
0	28,51	57,47
0,2	25,08	51,07
0,4	25,23	50,02
0,6	27,30	51,93
0,8	26,19	48,66
1	27,03	48,07

Fonte: elaborado pelo autor

Figura 13: Relação entre a média dos gaps obtidos e o ρ utilizado na heurística construtiva



Fonte: elaborado pelo autor

Na Tabela 3 e Tabela 4 encontramos os resultados obtidos utilizando a heurística construtiva e heurística de melhoria, respectivamente. Com a heurística construtiva chegamos em um *gap* médio da energia em comparação a solução exata de 25,34% e um *gap* médio do atraso de 54,25%. Já para heurística de melhoria, que utiliza a anterior como ponto de partida, foi possível alcançar um *gap* médio da energia de 7,49% e um *gap* médio do atraso 21,84%. Podemos notar que a heurística de melhoria foi capaz de diminuir em aproximadamente 18 pontos percentuais do *gap de* consumo de energia e 32 pontos percentuais do *gap de* atraso total, e mesmo com um aumento no custo computacional, isso torna-se insignificante quando comparado à melhoria nos resultados alcançados.

A Tabela 5 mostra a comparação dos tempos de execução entre os métodos explorados neste trabalho, o método exato e os dois métodos heurísticos. É possível notar que o tempo computacional necessário para os métodos heurísticos, com médias de 0,0027 e 3,2911 segundos, é consideravelmente menor quando comparado ao método utilizado pelo modelo matemático, que levou em média 944,26 segundos.

Tabela 3: Resultados computacionais utilizando a heurística construtiva

Tarefas	**Máquinas**	**Instância**	**Heurística construtiva**			
			Energia (UE)	**Atraso Total (UT)**	**GAP Energia (%)**	**GAP Atraso (%)**
5	5	1	755	908	18,71	-16,62
5	5	2	799	1874	21,61	40,59
5	5	3	745	1347	17,88	46,89
6	5	1	783	1611	17,74	41,94
6	5	2	831	1118	26,48	35,02
6	5	3	672	1502	22,18	73,44
7	5	1	777	1696	32,37	89,50
7	5	2	952	2453	25,43	74,34
7	5	3	845	3157	32,65	120,00
8	5	1	1043	3004	28,77	95,57
8	5	2	833	1623	12,57	45,04
8	5	3	767	2035	25,94	65,18
5	6	1	891	1590	33,98	56,65
5	6	2	833	1256	12,11	42,08

5	6	3	884	2023	19,78	81,43
6	6	1	982	2253	20,05	77,26
6	6	2	978	1762	14,25	53,22
6	6	3	876	2198	7,48	27,72
7	6	1	1017	3160	21,36	90,82
7	6	2	990	2188	29,75	38,66
7	6	3	1121	1152	29,15	-6,42
8	6	1	1075	1880	27,82	13,73
8	6	2	1003	2598	34,99	81,05
8	6	3	1151	4591	54,29	111,86
5	7	1	948	1506	25,40	46,78
5	7	2	829	1412	14,34	19,66
5	7	3	750	1280	21,56	132,30
6	7	1	1006	2620	19,62	18,23
6	7	2	1109	3258	24,75	95,09
6	7	3	913	2733	24,73	66,54
7	7	1	965	2675	22,31	5,23
7	7	2	973	3146	33,47	77,64
7	7	3	981	2379	27,40	13,39
8	7	1	982	3557	15,12	30,58
8	7	2	1262	2634	33,54	29,12
8	7	3	1120	3173	23,89	78,86
7	8	1	1244	2140	28,51	67,71
7	8	2	1152	3208	23,08	62,35
7	8	3	1189	2248	31,96	61,84
8	8	1	1265	3463	34,15	53,71
8	8	2	1251	3534	23,98	21,19
8	8	3	1287	2354	20,39	10,10
9	8	1	1334	4029	34,34	49,94
9	8	2	1475	4073	45,90	39,68
9	8	3	1269	4312	24,53	82,17
			Média		25,34	54,25

Fonte: elaborado pelo autor

Tabela 4: Resultados computacionais utilizando a heurística de melhoria

Tarefas	Máquinas	Instância	Heurística de melhoria			
			Energia (UE)	Atraso Total (UT)	GAP Energia (%)	GAP Atraso (%)
5	5	1	646	1365	1,57	25,34
5	5	2	672	1551	2,28	16,35
5	5	3	669	1192	5,85	29,99
6	5	1	677	1345	1,80	18,50
6	5	2	686	854	4,41	3,14
6	5	3	567	876	3,09	1,15
7	5	1	642	1368	9,37	52,85
7	5	2	795	1637	4,74	16,35
7	5	3	663	1923	4,08	34,01
8	5	1	831	1640	2,59	6,77
8	5	2	786	1177	6,22	5,18
8	5	3	669	1581	9,85	28,33
5	6	1	735	953	10,53	-6,11
5	6	2	784	1065	5,52	20,48
5	6	3	830	1675	12,47	50,22
6	6	1	830	1468	1,47	15,50
6	6	2	875	1297	2,22	12,78
6	6	3	826	2007	1,35	16,62
7	6	1	878	1802	4,77	8,82
7	6	2	792	1923	3,80	21,86
7	6	3	924	1556	6,45	26,40
8	6	1	920	1910	9,39	15,55
8	6	2	790	1674	6,33	16,66
8	6	3	823	2354	10,32	8,63
5	7	1	857	1397	13,36	36,16
5	7	2	800	1647	10,34	39,58
5	7	3	749	1280	21,39	132,30
6	7	1	896	2413	6,54	8,89

6	7	2	957	2211	7,65	32,40
6	7	3	761	1687	3,96	2,80
7	7	1	888	3059	12,55	20,34
7	7	2	796	2145	9,19	21,12
7	7	3	815	2609	5,84	24,36
8	7	1	914	2922	7,15	7,27
8	7	2	1021	2119	8,04	3,87
8	7	3	989	2266	9,40	27,73
7	8	1	1053	1344	8,78	5,33
7	8	2	980	2220	4,70	12,35
7	8	3	972	2328	7,88	67,60
8	8	1	1018	2685	7,95	19,17
8	8	2	1120	2663	11,00	-8,68
8	8	3	1164	2911	8,89	36,16
9	8	1	1150	3528	15,81	31,30
9	8	2	1197	3155	18,40	8,20
9	8	3	1097	2582	7,65	9,08
			Média		7,49	21,84

Fonte: elaborado pelo autor

Tabela 5: Tempo computacional para encontrar soluções utilizando os 3 diferentes métodos

Tarefas	Máquinas	Instância	Tempo computacional (segundos)		
			Modelo matemático	Heurística construtiva	Heurística de melhoria
5	5	1	33	0,0030	0,1654
5	5	2	8	0,0020	0,1656
5	5	3	10	0,0010	0,1506
6	5	1	14	0,0020	0,4104
6	5	2	23	0,0010	0,6572
6	5	3	90	0,0020	0,5658
7	5	1	25	0,0020	1,3105
7	5	2	99	0,0020	7,6387
7	5	3	36	0,0020	1,3887
8	5	1	77	0,0020	2,3772
8	5	2	326	0,0020	1,8581
8	5	3	275	0,0020	1,7600
5	6	1	843	0,0020	0,3920
5	6	2	232	0,0030	0,2554
5	6	3	647	0,0020	0,3072
6	6	1	1800	0,0020	0,6717
6	6	2	1800	0,0020	0,5376
6	6	3	98	0,0020	0,5037
7	6	1	601	0,0030	1,8672
7	6	2	900	0,0040	2,1126
7	6	3	744	0,0020	1,2178
8	6	1	253	0,0030	2,1644
8	6	2	899	0,0010	5,0979
8	6	3	1106	0,0020	4,3812
5	7	1	258	0,0040	0,2982
5	7	2	745	0,0010	0,3241
5	7	3	1254	0,0020	0,2214
6	7	1	222	0,0050	1,2276
6	7	2	1800	0,0020	0,8753

6	7	3	1800	0,0020	0,9159
7	7	1	305	0,0020	2,4951
7	7	2	1800	0,0020	1,4072
7	7	3	1800	0,0030	1,6885
8	7	1	1800	0,0060	6,4931
8	7	2	1800	0,0030	3,5745
8	7	3	1800	0,0030	2,9483
7	8	1	1800	0,0040	4,1308
7	8	2	1800	0,0030	1,8144
7	8	3	1800	0,0040	4,5052
8	8	1	1800	0,0060	11,0913
8	8	2	1800	0,0040	4,4713
8	8	3	1800	0,0030	4,1835
9	8	1	1800	0,0060	14,3794
9	8	2	1800	0,0030	20,8115
9	8	3	1800	0,0040	22,2855
Média			944,96	0,0027	3,2911

Fonte: elaborado pelo autor

6.2.3. Análise de Sensibilidade

Para realizarmos esta análise utilizaremos as instâncias geradas para 5, 6, 7 e 8 tarefas em ambientes de 5 máquinas. Para isso, foram criados dois cenários, partindo do cenário base que encontramos os resultados na Tabela 2, Tabela 3 e Tabela 4. Estes dois cenários terão um aumento no tamanho da janela de indisponibilidade de 10% e 20% respectivamente. As instâncias foram então resolvidas, utilizando o método exato e os métodos heurísticos, o *gap* médio da energia total gasta e o *gap* médio do atraso total obtido comparando estes métodos podem ser encontrados na Tabela 6. Podemos notar que tanto a heurística construtiva, quanto a heurística de melhoria apresentam valores percentuais nos cenários com aumento da janela de indisponibilidade, para o *gap* da energia, semelhantes ou até menores daquele que foi obtido no cenário base. Isso demonstra que os métodos heurísticos são capazes de lidar com cenários em que as máquinas possuem mais restrições de disponibilidade. Porém em ambos os métodos houve um aumento no *gap* do atraso total encontrado, representando que os

valores encontrados para o atraso total das tarefas foi mais distante daquele obtido pelo método exato com o aumento das janelas de indisponibilidade.

Tabela 6: *Gaps* médio da energia total gasta e atraso total para cenários com aumento na janela de indisponibilidade

	Heurística construtiva		**Heurística de melhoria**	
Janela de Indisponibilidade	**GAP Energia (%)**	**GAP Atraso (%)**	**GAP Energia (%)**	**GAP Atraso (%)**
Base	23,53	59,24	4,66	19,83
Com aumento de 10%	22,25	64,33	3,75	25,35
Com aumento de 20%	22,51	65,61	4,22	22,41

Fonte: elaborado pelo autor

6.2.4. Instâncias de grande escala

Na Tabela 7 e Tabela 8 temos os resultados obtidos utilizando a heurística construtiva e a heurística de melhoria, respectivamente, nas instâncias com maior escala, que também representam cenários que podem ser encontrados na prática. A heurística construtiva apresentou uma média do tempo computacional de 0,34 segundos. Já a heurística de melhoria, com o aumento do número de tarefas apresentou um crescimento no tempo computacional necessário para aplicação do método, mas apesar deste aumento ainda apresenta tempos aceitáveis para uma aplicação prática, ficando com uma média de 860,30 segundos. Isso se deve ao fato da vizinhança explorada crescer com o aumento do número de tarefas, para o caso de 200 tarefas o tempo computacional já atingiu o limite de 30 minutos aplicado, isso demonstra que com o aumento da escala da instância pode ser importante explorar outros tipo de movimento gerado pela busca local, para que seja possível explorar soluções mais vantajosas em um tempo menor, sendo que, o movimento utilizado neste trabalho apresenta uma maior vantagem para instâncias menores.

Tabela 7: Resultados obtido com a heurística construtiva nas instâncias de maior escala

Tarefas	Máquinas	Instância	Heurística construtiva		
			Energia (UE)	Atraso Total (UT)	Tempo computacional (segundos)
20	10	1	3918	11994	0,0160
20	10	2	3474	15234	0,0090
20	10	3	2790	14180	0,0090
50	10	1	6228	51097	0,0539
50	10	2	4621	28454	0,0558
50	10	3	4835	39960	0,0621
100	10	1	11182	83173	0,3779
100	10	2	7940	74571	0,2329
100	10	3	9988	153476	0,2199
200	10	1	15956	251033	0,8912
200	10	2	20577	166923	0,9909
200	10	3	20397	105074	1,1818
			Média		0,34

Fonte: elaborado pelo autor

Tabela 8: Resultados obtido com a heurística de melhoria nas instâncias de maior escala

Tarefas	Máquinas	Instância	Heurística de Melhoria		
			Energia (UE)	Atraso Total (UT)	Tempo computacional (segundos)
20	10	1	2058	15423	81,61
20	10	2	2286	14639	197,87
20	10	3	2002	12197	61,89
50	10	1	4787	65090	409,51
50	10	2	4257	32750	410,26
50	10	3	4447	53025	411,89
100	10	1	9411	128727	1299,91
100	10	2	7629	81962	1314,72
100	10	3	8689	173023	1335,98
200	10	1	14503	263894	1600
200	10	2	17763	177270	1600
200	10	3	19438	110026	1600
			Média		860,30

Fonte: elaborado pelo autor

7. CONCLUSÃO

O presente trabalho teve como objetivo explorar o problema de minimização de consumo de energia e atraso total no sequenciamento de tarefas *non-resumables* em um ambiente *flow shop* com períodos de indisponibilidade. Para isso, com base em conceitos aprendidos no curso de Engenharia de Produção podemos contextualizar e descrever características que nos ajudam a entender particularidades desse problema. O referencial teórico retirado de estudos acadêmicos foi de extrema importância para guiar o andamento do trabalho e demonstrar lacunas para contribuição, principalmente por se tratar de um problema ainda pouco endereçado ao conhecimento do autor.

O tema estudado reforça a relevância de integrar práticas sustentáveis à gestão operacional. Os resultados obtidos demonstram que é possível alinhar eficiência produtiva com a redução de impactos ambientais, contribuindo diretamente para os Objetivos de Desenvolvimento Sustentável (ODS) da ONU, especialmente os ODS 9 (Indústria, Inovação e Infraestrutura) e 12 (Consumo e Produção Responsáveis). Assim, este estudo não apenas atende à demanda por processos mais otimizados, mas também enfatiza a importância de estratégias que promovam a sustentabilidade e a responsabilidade corporativa no setor industrial.

Utilizando como base Wilson (1989) e Meng *et al.* (2019) foi desenvolvido um modelo matemático que busca encontrar o melhor sequenciamento de tarefas, otimizado de maneira lexicográfica, isto é, atribuindo maior prioridade para a minimização do consumo energético em comparação à minimização do atraso total. Para o controle do consumo de energia foi atribuída ao modelo a decisão de utilizar a estratégia de ligar e desligar máquinas durante períodos ociosos entre tarefas.

Devido a complexidade do problema, que assim como muitos problemas de *scheduling* apresenta caráter combinatório *NP-hard* , foram propostos métodos heurísticos que não garantem a solução ótima, mas apresentam soluções satisfatórias em tempo viável. A heurística construtiva apresentada utilizou das características do problema para sequenciar as tarefas, fazendo o aproveitamento de janelas para diminuir o tempo ocioso, além de levar em consideração a data de entrega das tarefas. Já a heurística de melhoria aproveitou o resultado obtido pela construtiva para, por meio de buscas locais e perturbações, encontrar melhores soluções, a metodologia utilizada para essa heurística apesar de não ser exatamente igual, se assemelha a ILS (*Iterated Local Search*) que se baseia em aprimorar soluções locais através

de um processo iterativo, sendo o aprimoramento para esta meta-heurística uma possibilidade para futuros trabalhos.

Observando os resultados obtidos, podemos concluir que com os métodos heurísticos foi possível obter resultados satisfatórios em tempos computacionais viáveis, apesar da heurística construtiva apresentar resultados um pouco mais distantes dos resultados ótimos, pode-se utilizar a mesma como uma solução inicial para a heurística de melhoria, que conseguiu atingir resultados mais satisfatórios com *gaps* médios do consumo total de energia e atraso total de 7,49% e 21,84% respectivamente, para os 45 experimentos utilizados na validação.

Outro ponto importante de se notar é que o *gap* de atraso total foi mais alto quando comparado ao consumo total de energia, isso se dá devido a relação que esses dois objetivos tem quanto a otimização. Muitas vezes o ajuste na programação das tarefas, ao beneficiar um dos objetivos, pode por vezes acarretar prejuízos ao outro. Como na construção desse problema o objetivo principal a ser otimizado foi o consumo de energia, os métodos utilizados foram desenhados com maior foco no mesmo, seja no modelo matemático, com a otimização lexicográfica, ou nos métodos heurísticos, com o pós-processamento.

Como direcionamento para pesquisas futuras, uma análise utilizando outras formas de otimização multi-objetivo poderia nos permitir entender melhor a relação entre os dois objetivos aqui tratados. O desenvolvimento das heurísticas propostas, como por exemplo, diferentes regras de geração de solução e análise de diferentes tipos de vizinhança, para que se adequem cada vez mais ao aumento da escala de instâncias. Além disso a extensão do problema para tarefas *resumables* e desta forma explorando a indisponibilidade das máquinas para os dois casos apresentados na literatura.

8. REFERÊNCIAS

AARTS, E.; LENSTRA, J. K. Local search in combinatorial optimization. Princeton: Princeton University Press, 2003.

ADIRI, I.; BRUNO, J.; FROSTIG, E.; RINNOOY KAN, A. H. G. Single machine flowtime scheduling with a single breakdown. Acta Informatica, v. 26, p. 679–696, 1989.

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

AGGOUNE, R.; PORTMANN, M.-C. Flow shop scheduling problem with limited machine availability: A heuristic approach. International Journal of Production Economics, v. 99, p. 4–15, 2006.

ALLAOUI, A.; ARTIBA, A.; ELMAGHRABY, S. E.; RIANE, F. Scheduling of a two-machine flowshop with availability constraints on the first machine. International Journal of Production Economics, v. 99, p. 16–27, 2006.

ARMENTANO, V. A.; RONCONI, D. P. Tabu search for total tardiness minimization in flowshop scheduling problems. Computers & Operations Research, v. 26, n. 3, p. 219–235, 1999.

ASSIA, S.; IKRAM, E. A.; BARKANY, A.; AHMED, E. B. Non-permutation flow shop scheduling problems with unavailability constraints to minimize total energy consumption. In: 5th International Conference on Optimization and Applications (ICOA), Kenitra, Morocco, 2019. p. 1–5.

BŁAZEWICZ, J.; BREIT, J.; FORMANOWICZ, P.; KUBIAK, W.; SCHMIDT, G. Heuristic algorithms for the two-machine flowshop with limited machine availability. Omega, v. 29, p. 599–608, 2001.

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

CUI, W.; LU, B. A bi-objective approach to minimize makespan and energy consumption in flow shops with peak demand constraint. Sustainability, v. 12, p. 4110, 2020.

COELLO, C. A. C. A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems, v. 1, n. 3, p. 269–308, 1999.

FANG, K.; UHAN, N. A.; ZHAO, F.; ZHANG, J. Flow shop scheduling with peak power consumption constraints. Annals of Operations Research, v. 206, p. 115–145, 2013.

FOUMANI, M.; SMITH-MILES, K. The impact of various carbon reduction policies on green flowshop scheduling. Applied Energy, v. 249, p. 300–315, 2019.

GIRET, A.; TRENTESAUX, D.; PRABHU, V. Sustainability in manufacturing operations scheduling: A state of the art review. Journal of Manufacturing Systems, v. 37, parte 1, p. 126–140, 2015.

ISERMANN, H. Linear lexicographic optimization. OR Spektrum, v. 4, n. 4, p. 223–228, 1982.

KUBIAK, W.; BŁAZEWICZ, J.; FORMANOWICZ, P.; BREIT, J.; SCHMIDT, G. Two-machine flow shops with limited machine availability. European Journal of Operational Research, v. 136, p. 528–540, 2002.

KUBZIN, M. A.; POTTS, C. N.; STRUSEVICH, V. A. Approximation results for flowshop scheduling problems with machine availability constraints. Computers & Operations Research, v. 36, p. 379–390, 2009.

LEE, C. Y. Machine scheduling with availability constraints. Journal of Global Optimization, v. 9, p. 363–382, 1996.

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

LEE, C. Y. Two-machine flowshop scheduling with availability constraints. European Journal of Operational Research, v. 114, p. 420–429, 1999.

LEE, J. Y.; KIM, Y. D. Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraint on the first machine. Computers & Industrial Engineering, v. 114, p. 22–30, 2017. DOI: 10.1016/j.cie.2017.10.004.

LI, M.; WANG, G. G. A review of green shop scheduling problem. Information Sciences, v. 589, p. 478–496, 2022. DOI: 10.1016/j.ins.2021.12.122.

MA, Y.; CHU, C.; ZUO, C. A survey of scheduling with deterministic machine availability constraints. Computers and Industrial Engineering, v. 58, n. 2, p. 199–211, 2010.

MANSOURI, S. A.; AKTAS, E.; BESIKCI, U. Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. European Journal of Operational Research, v. 248, n. 3, p. 772–788, 2016.

MENG, Leilei; ZHANG, Chaoyong; SHAO, Xinyu; REN, Yaping. MILP models for energy-aware flexible job shop scheduling problem. Journal of Cleaner Production, v. 210, p. 710–723, 2019.

MOUZON, G.; YILDIRIM, M. B.; TWOMEY, J. Operational methods for minimization of energy consumption of manufacturing equipment. International Journal of Production Research, v. 45, n. 18–19, p. 4247–4271, 2007.

MOUZON, G.; YILDIRIM, M. B. A framework to minimize total energy consumption and total tardiness on a single machine. International Journal of Sustainable Engineering, v. 1, n. 2, p. 105–116, 2008. DOI: 10.1080/19397030802257236.

NICHOLSON, T. A. J. Optimization in industry: Optimization techniques. London Business School series. Chicago: Aldine Atherton, 1971.

PINEDO, M. L. Scheduling: Theory, Algorithms, and Systems. New York: Springer, 2016.

RONCONI, D. P.; HENRIQUES, L. R. S. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. Omega, v. 37, p. 272–281, 2009.

SADFI, C.; PENZ, B.; RAPINE, C.; BŁAZEWICZ, J.; FORMANOWICZ, P. An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. European Journal of Operational Research, v. 161, p. 3–10, 2005.

TAILLARD, E. Benchmarks for basic scheduling problems. European Journal of Operational Research, v. 64, n. 2, p. 278–285, 1993.

WILSON, J. M. Alternative formulations of a flow-shop scheduling problem. Journal of the Operational Research Society, v. 40, n. 4, p. 395–399, 1989.

ZANAKIS, S. H.; EVANS, J. R.; VAZACOPOULOS, A. A. Heuristic methods and applications: A categorized survey. European Journal of Operational Research, v. 43, p. 88–110, 1989.

ANEXO A - Código do modelo matemático utilizando Gurobi na linguagem Python

```
import gurobipy as gp
q = gp.Model()

#Inserir variáveis de decisão
x = q.addVars(jobs,pos, vtype = gp.GRB.BINARY)
S = q.addVars(maquinas,jobs, vtype = gp.GRB.CONTINUOUS)
F = q.addVars(maquinas,jobs, vtype = gp.GRB.CONTINUOUS)
T = q.addVars(jobs, vtype = gp.GRB.CONTINUOUS)
Z = q.addVars(maquinas,jobs, vtype = gp.GRB.BINARY)
U = q.addVars(maquinas,jobs, vtype = gp.GRB.CONTINUOUS)
W = q.addVars(maquinas,jobs, vtype = gp.GRB.CONTINUOUS)

q.setParam('TimeLimit', 30 * 60) #Limite 30 minutos
q.setParam("IntegralityFocus",1)

#Definir Função objetivo
q.setObjective(gp.quicksum(((U[r,j+1] - W[r,j]) * Pidle[r] + EnergyS[r] * Z[r,j]) for r in maquinas for j in range(n+m-1)) + P0*F[m-1,n+m-1] + (gp.quicksum(T[j] for j in jobs)/UB), sense=gp.GRB.MINIMIZE)

#Restrição 5
c1 = q.addConstrs(
   gp.quicksum(x[i,j] for i in jobs) == 1 for j in pos
)

#Restrição 6
c2 = q.addConstrs(
   gp.quicksum(x[i,j] for j in pos) == 1 for i in jobs
)

#Restrição 7
c3 = q.addConstrs(
   S[r,j] - s[r] <= M * (1 - x[n+r,j]) for r in maquinas for j in pos
)

#Restrição 8
c4 = q.addConstrs(
   S[r,j] - s[r] >= - M * (1 - x[n+r,j]) for r in maquinas for j in pos
```

```
)

#Restrição 9
c5 = q.addConstrs(
     S[r,j+1] >= S[r,j] + gp.quicksum(p[r,i]*x[i,j] for i in jobs) for r in maquinas for j in
range(n+m-1)
)

#Restrição 10
c6 = q.addConstrs(
   S[r+1,j] >= S[r,j] + gp.quicksum(p[r,i]*x[i,j] for i in jobs) for r in range(m-1) for j in pos
)

#Restrição 11
c7 = q.addConstr(
   S[0,0] >= 0
)

#Restrição 12
c8 = q.addConstrs(
   T[j] >= S[m-1,j]+gp.quicksum(x[i,j]*(p[m-1,i]-d[i]) for i in jobs) for j in pos
)

#Restrição 13
c9 = q.addConstrs(
   T[j] >= 0 for j in pos
)

#Restrição 14

c10 = q.addConstrs(
   F[r,j] == S[r,j]+gp.quicksum(p[r,i]*x[i,j] for i in jobs) for r in maquinas for j in pos
)

#Restrição 15

c11 = q.addConstrs(
   S[r,j+1] - F[r,j] >= TB[r] - M * (1 - Z[r,j]) for r in maquinas for j in range(n+m-1)
)

#Restrição 16

c11 = q.addConstrs(
   S[r,j+1] - F[r,j] <= TB[r] + M * Z[r,j] for r in maquinas for j in range(n+m-1)
```

```
)

#Restrição 17
c12 = q.addConstrs(
   gp.quicksum(Z[r,j] for j in range(n+m-1)) <= Nes[r] for r in maquinas
)

#Restrição 18

c13 = q.addConstrs(
   U[r,j+1] >= S[r,j+1] - M * Z[r,j] for r in maquinas for j in range(n+m-1)
)

#Restrição 19

c14 = q.addConstrs(
   U[r,j+1] <= S[r,j+1] + M * Z[r,j] for r in maquinas for j in range(n+m-1)
)

#Restrição 20

c15 = q.addConstrs(
   U[r,j+1] <=  M * (1 - Z[r,j]) for r in maquinas for j in range(n+m-1)
)

#Restrição 21

c16 = q.addConstrs(
   U[r,j] >= 0 for r in maquinas for j in pos
)

#Restrição 22

c17 = q.addConstrs(
   W[r,j] >= F[r,j] - M * Z[r,j] for r in maquinas for j in range(n+m-1)
)

#Restrição 23

c18 = q.addConstrs(
   W[r,j] <= F[r,j] + M * Z[r,j] for r in maquinas for j in range(n+m-1)
)

#Restrição 24
```

```
c19 = q.addConstrs(
    W[r,j] <=  M * (1 - Z[r,j]) for r in maquinas for j in range(n+m-1)
)

#Restrição 25

c20 = q.addConstrs(
    W[r,j] >= 0 for r in maquinas for j in pos
)

q.optimize()
```

ANEXO B - Código da heurística construtiva e de melhoria em Python

```
import time
import random
import math

def carregar_instancia(arquivo_caminho, numero_instancia,x):
    with open(arquivo_caminho, 'r') as f:
        linhas = f.readlines()

    instancias = []
    instancia_atual = []

    for linha in linhas:
        linha = linha.strip()
        if linha == str(x):
            if instancia_atual:
                instancias.append(instancia_atual)
                instancia_atual = []
        else:
            numeros = list(map(int, linha.split()))
            instancia_atual.append(numeros)

    if instancia_atual:  # Adiciona a última instância
        instancias.append(instancia_atual)

    if 1 <= numero_instancia <= len(instancias):
        return instancias[numero_instancia - 1]  # Retorna a instância escolhida
    else:
        raise ValueError(f"Número de instância inválido. Existem {len(instancias)} instâncias.")

def soma_parcial(instancia, x):
    vetor_somas = []
      sobra_acumulada = []  # Para acumular os números que não foram usados nas linhas anteriores

    for linha in instancia:
        soma_primeiros_x = round(sum(linha[:x])/2)

        soma_com_sobra_acumulada = soma_primeiros_x + sum(sobra_acumulada)

        vetor_somas.append(soma_com_sobra_acumulada)
```

```
        sobra_acumulada.extend(linha[x:])  # Acumula os números após os primeiros 'x'

    return vetor_somas

def min_positive(a, b):
    return min(x for x in (a, b,100000) if x >= 0)

def valoresD(job):
#Calcula o valor de D para o primeiro job
    if len(ordem)==0:
            acumulado = 0
    for chave,valor in D.items():
        x,y = chave
        if y == job:
            if x == 0 and len(ordem)>0:
                acumulado = D[x,ordem[-1]]
            if x>0 and len(ordem)>0:
                if n+x in maqindisp:
                    acumulado = max(D[x,ordem[-1]],D[x-1,y],s[x]+p[x,n+x])
                else:
                    acumulado = max(D[x,ordem[-1]],D[x-1,y])
            S[chave] = acumulado
            if S[chave] <= s[x] and S[chave] + p[chave] > s[x]:
                S[chave] = s[x]+p[x,n+x]
                S[x,n+x] = s[x]
                D[x,n+x] = s[x]+p[x,n+x]
                S[chave] = max(acumulado, s[x]+p[x,n+x])
                if n+x not in ordem:
                    ordem3.append(n+x)
                maqindisp.append(x)
            if S[chave] > s[x] and n+x not in ordem3:
                S[x,n+x] = s[x]
                D[x,n+x] = s[x]+p[x,n+x]
                if len(ordem)==0:
                    S[chave] = max(D[x-1,y],s[x]+p[x,n+x])
                else:
                    S[chave] = max(D[x,ordem[-1]],D[x-1,y],s[x]+p[x,n+x])
                if n+x not in ordem:
                    ordem3.append(n+x)
                maqindisp.append(x)

            acumulado = S[chave] + p[chave]
            D[chave] = acumulado
```

```
def gerar_vizinhanca_insercao(x):
    vizinhanca = set()
    n = len(x)

    for i in range(n):
        # Remove o elemento x[i]
        for j in range(n):
            if i != j:
                nova_permutacao = list(x)
                elemento = nova_permutacao.pop(i)  # Remove o elemento na posição i
                nova_permutacao.insert(j, elemento)  # Insere o elemento na posição j
                vizinhanca.add(tuple(nova_permutacao))

    return vizinhanca

def posproc(D,S,ordem3): #Pós-processamento
    ordem2=[]
    for job in reversed(ordem3):
        if job < n:
            ordem2.append(job)
    for maqs, teste in reversed(p_maquinas.items()):
        if maqs>=0:
            ordemtemp = [job for job in ordem2 if job < n ]
            #print(ordemtemp)
            for i in range(n):
                if i > 0:
                    if D[maqs,ordem2[i]] > s[maqs]: #+ p[maqs,n+maqs)
                        if maqs == m-1:
                            D[maqs,ordem2[i]] = S[maqs,ordem2[i-1]]
                        else:
                            D[maqs,ordem2[i]] = min(S[maqs,ordem2[i-1]], S[maqs+1,ordem2[i]])
                        S[maqs,ordem2[i]] = D[maqs,ordem2[i]] - p[maqs,ordem2[i]]
                    if D[maqs,ordem2[i]] <= s[maqs]:
                        if maqs == m-1:
                            if S[maqs,ordem2[i-1]] - D[maqs,n+maqs] >= p[maqs,ordem2[i]]:
                                D[maqs,ordem2[i]] = S[maqs,ordem2[i-1]]
                                indice = ordem3.index(ordem2[i])
                                ordem3.remove(n+maqs)
                                ordem3.insert(indice,n+maqs)
                                S[maqs,ordem2[i]] = D[maqs,ordem2[i]] - p[maqs,ordem2[i]]
                        if maqs != m-1:
                            if min(S[maqs,ordem2[i-1]], S[maqs+1,ordem2[i]]) - D[maqs,n+maqs] >=
p[maqs,ordem2[i]]:
```

```
                    D[maqs,ordem2[i]] = min(S[maqs,ordem2[i-1]], S[maqs+1,ordem2[i]])
                    indice = ordem3.index(ordem2[i])
                    ordem3.remove(n+maqs)
                    ordem3.insert(indice,n+maqs)
                else:
                                                        D[maqs,ordem2[i]] =
min(S[maqs,ordem2[i-1]],s[maqs],S[maqs+1,ordem2[i]])
                    S[maqs,ordem2[i]] = D[maqs,ordem2[i]] - p[maqs,ordem2[i]]
                if D[maqs,ordem2[i]] <= s[maqs]:
                    if maqs == m-1:
                        D[maqs,ordem2[i]] = min(S[maqs,ordem2[i-1]],s[maqs])
                    else:
                                                D[maqs,ordem2[i]] = min(S[maqs,ordem2[i-1]],
S[maqs+1,ordem2[i]],s[maqs])
                    S[maqs,ordem2[i]] = D[maqs,ordem2[i]] - p[maqs,ordem2[i]]

def FO(D,S,ordem3):
    ordemoriginal = ordem3.copy()
    atrasotot = 0
    for i in ordem:
        atraso = max(0,(D[m-1,i])-d[i])
        atrasotot = atrasotot + atraso

    energia=0
    for r in range(m):
        contoff = 0
        ordemtemp = [job for job in ordemoriginal if job < n or job == n+r]
        for i in range(len(ordemtemp)-1):
            if S[r,ordemtemp[i+1]] - D[r,ordemtemp[i]] > TB[r] and contoff<Nes[r]:
                energia = energia + EnergyS[r]
                contoff = contoff + 1
            else:
                energia = energia + ((S[r,ordemtemp[i+1]] - D[r,ordemtemp[i]])*Pidle[r])
    energia = energia + P0*(D[m-1,ordem[-1]])

    return energia,atrasotot

def troca_aleatoria(lista, x):
    for _ in range(x):
        # Escolhe duas posições aleatórias diferentes na lista
        i, j = random.sample(range(len(lista)), 2)
        # Troca os elementos nas posições i e j
        lista[i], lista[j] = lista[j], lista[i]
    return lista
```

```
n = 9
m = 8
nome = f"{n:02}-{m:02}.txt"
nome1 = f"{n:02}-{m:02}d.txt"
instancias = [1,2,3]
t = n+m
qtd_jobs = range(n+m)
qtd_maquinas = range(m)
print("Arquivo:",nome)

for instancia in instancias:
    vet_p = carregar_instancia(nome, instancia, m)
    vet_d = carregar_instancia(nome1,instancia,n)[0]

    mind=min(vet_d)
    somap = [sum(lista) for lista in vet_p]
    UB = (sum(somap)-mind)*(m+n)
    M = UB
    for i in range(m):
        vet_d.append(M)
    vet_s = soma_parcial(vet_p, n)

    #Energia gasta em tempo IDLE
    vet_Pidle = []
    #Energia gasta pela fábrica
    P0 = 1
    #Energia gasta para ligar e desligar
    vet_EnergyS = []
    vet_Nes = [] #Numero máximo de vezes que uma maquina para desligar e ligar
    vet_TB = [] #Tempo de Break-even da máquina

    for i in range(m):
        vet_Pidle.append(2)
        vet_EnergyS.append(1)
        vet_Nes.append(3)
        vet_TB.append(20)

    #Rótulos jobs
    jobs = list(qtd_jobs)

    #Rótulos posições
    pos = list(qtd_jobs)
```

```
#Rótulos máquinas
maquinas = list(qtd_maquinas)

#Dicionário dos tempos de processamento nas máquinas
p = dict()
for i in range(n+m):
    for r in range(m):
        p[r,i]=vet_p[r][i]

#Dicionário inicio do periodo de indisponibilidade nas maquinas
s = dict()
for i in range(m):
    s[i] = vet_s[i]

d = dict()
for i in range(n+m):
    d[i] = vet_d[i]

D = dict()
for i in range(n+m):
    for r in range(m):
        D[r,i]=0

S = dict()
for i in range(n+m):
    for r in range(m):
        S[r,i]=0

Pidle = dict()
for i in range(m):
    Pidle[i] = vet_Pidle[i]

EnergyS = dict()
for i in range(m):
    EnergyS[i] = vet_EnergyS[i]

Nes = dict()
for i in range(m):
    Nes[i] = vet_Nes[i]

TB = dict()
for i in range(m):
    TB[i] = vet_TB[i]
```

```
jobs = [i for i in range(n)]
ordem = []
ordem3 = []
ps = p
ds = d
for i in range(n,n+m):
    ps = {k: v for k, v in ps.items() if k[1] != i}
    ds = {k: v for k, v in ds.items() if k != i}

ps_maquinas = {}
p_maquinas = {}

# Iterar sobre o dicionário original
for chave, valor in ps.items():
    x, y = chave  # Separar os elementos da chave (x, y)
    # Se o valor y não estiver no novo dicionário, criar um dicionário para ele
    if x not in ps_maquinas:
        ps_maquinas[x] = {}
    # Adicionar a chave (x, y) e seu valor no dicionário correspondente a y
    ps_maquinas[x][chave] = valor

# Iterar sobre o dicionário original
for chave, valor in p.items():
    x, y = chave  # Separar os elementos da chave (x, y)
    # Se o valor y não estiver no novo dicionário, criar um dicionário para ele
    if x not in p_maquinas:
        p_maquinas[x] = {}
    # Adicionar a chave (x, y) e seu valor no dicionário correspondente a y
    p_maquinas[x][chave] = valor

I=1000000
for chave,valor in ps_maquinas[0].items():
    x,y = chave
    if valor + ds[y]< I:
        I = valor + ds[y]
        job = y

valoresD(job)
ordem.append(job)
ordem3.append(job)
jobs.remove(job)

maqindisp=[]
```

```
   F=[]
   flag = 0
   rho = 0.5
   while jobs:
      Fref = 1000000
      flag = 0
      for job in jobs:
         F = 0
         janela = 0
         LB = 0
         pj = 0
         for maqs, teste in ps_maquinas.items():
            pj = pj + ps[maqs,job]
            if maqs != m-1:
               if maqs ==0:
                        bj =  min_positive(D[maqs+1,ordem[-1]]-D[maqs,ordem[-1]], s[maqs] -
D[maqs,ordem[-1]])
               else:
                  pacumul=0
                  for r in range(maqs):
                     pacumul = pacumul+p[r,job]
                                                                                    bj     =
min_positive(D[maqs+1,ordem[-1]]-max(D[maqs,ordem[-1]],D[0,ordem[-1]]+pacumul),
s[maqs] - max(D[maqs,ordem[-1]],D[0,ordem[-1]]+pacumul) )
               janela = abs(bj-ps[maqs,job]) + janela
         LB = ds[job] - pj
         F = rho * (janela) + (1-rho)*(LB-D[0,ordem[-1]])
         if F<Fref:
            Fref = F
            jobref = job
            jobref1 = job
         if F==Fref and ps[0,job]>ps[0,jobref]:
            Fref = F
            jobref = job
            jobref1 = job

      valoresD(jobref)
      ordem.append(jobref)
      ordem3.append(jobref)
      if jobref in jobs:
         jobs.remove(jobref)

   posproc(D,S,ordem3)
   energia,atrasotot = FO(D,S,ordem3)
```

```
start_time = time.time()
ordemi = ordem
flag=1
flag1 =0
energiac = energia
atrasototc = atrasotot
energiai = energia
atrasototi = atrasotot
Di=D
Si=S
while flag1 < 10:
    while flag == 1:
        flag=0
        vizinhos = gerar_vizinhanca_insercao(ordemi)
        for v in vizinhos:
            for i in range(n+m):
                for r in range(m):
                    D[r,i]=0
            for i in range(n+m):
                for r in range(m):
                    S[r,i]=0
            ordem=[]
            ordem3=[]
            for job in v:
                valoresD(job)
                ordem3.append(job)
                ordem.append(job)
            posproc(D,S,ordem3)
            energia2,atrasotot2 = FO(D,S,ordem3)
            if energia2 < energiai:
                energiai = energia2
                atrasototi = atrasotot2
                Di=D
                Si=S
                ordemi = ordem
                flag = 1
                flag1=0
    ordemi = troca_aleatoria(ordemi,math.ceil(n/2))
    flag1 = flag1 + 1
    flag = 1

print("Cosntrutiva",energiac,atrasototc)
print("Melhoria",energiai,atrasototi)
end_time = time.time()  # Registra o horário final
```

```
execution_time= end_time - start_time  # Calcula o tempo de execução
numero_com_virgula = str(execution_time).replace('.', ',')
```