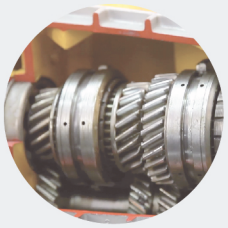
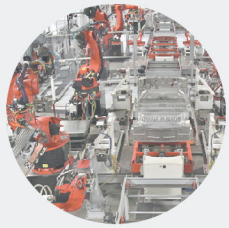
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA E GESTÃO INDUSTRIAL



PROBLEMA DE ESCALONAMENTO NUM SISTEMA FLEXIBLE JOB-SHOP

RENATA FILIPA MORIM CORREIA
outubro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

PROBLEMA DE ESCALONAMENTO NUM SISTEMA JOB-SHOP FLEXÍVEL

Renata Filipa Morim Correia

Dissertação para obtenção do Grau de Mestre em

Engenharia e Gestão Industrial

Orientador: Professora Doutora Alzira Maria Teixeira da Mota

Co-orientador: Professor Doutor Luís Adriano Preto Mendes Afonso

Júri:

Presidente:

Professor Doutor Manuel Joaquim Pereira Lopes, Professor Coordenador, Instituto Superior de Engenharia do Porto

Vogais:

Professora Doutora Ana Maria Pinto de Moura, Professora Auxiliar, Universidade de Aveiro

Professora Doutora Alzira Maria Teixeira da Mota, Professora Adjunto, Instituto Superior de Engenharia do Porto

Professor Doutor Luís Adriano Preto Mendes Afonso, Professor Adjunto, Instituto Superior de Engenharia do Porto

Porto, setembro 2024

Resumo

O presente trabalho estuda o escalonamento em sistemas *Job-Shop* flexíveis com divisão de lotes, motivado por um problema real da indústria têxtil. Inicialmente, foi apresentado um modelo de programação linear inteira mista, com o objetivo de minimizar o tempo de conclusão de todos os trabalhos, utilizando a técnica de *job-splitting*, que possibilita a divisão de um lote em sublotes para operações com múltiplas máquinas disponíveis. Este modelo, implementado em Python com a biblioteca Pulp e o solver CPLEX, mostrou-se viável para instâncias pequenas, mas revelou-se incapaz de encontrar soluções em tempos aceitáveis para instâncias de média dimensão.

Para superar esta limitação, é proposto um Algoritmo Genético Híbrido, que incorpora um algoritmo de pesquisa local para otimizar tanto o número quanto o tamanho dos sublotes. A meta-heurística, também desenvolvida em Python, foi testada em diversas instâncias, apresentando resultados promissores em termos de tempo de CPU. Em avaliações de instâncias reais da indústria têxtil, foi observado um tempo de CPU de aproximadamente 20 minutos para uma instância com 57 trabalhos e 35 minutos para uma com 67 trabalhos.

Adicionalmente, foi explorada a técnica de *lot streaming*, que permite a divisão de trabalhos em sublotes tratados de forma independente ao longo do processo produtivo. Um modelo de programação linear inteira mista foi apresentado e implementado, e, embora os resultados obtidos tenham superado os do *job-splitting*, este modelo mostrou-se inviável para algumas instâncias devido aos tempos computacionais.

Este trabalho destaca a importância das técnicas desenvolvidas e evidencia que a aplicação de meta-heurísticas pode facilitar a obtenção de soluções eficientes para problemas de escalonamento na indústria, em particular, no setor têxtil.

Palavras-chave: Escalonamento, *Job-Shop* Flexível, *lot streaming*, *job-splitting*, programação linear inteira mista, algoritmo genético híbrido

Abstract

The present work studies scheduling in flexible job shop systems with lot sizing, motivated by a real problem in the textile industry. First, a mixed-integer linear programming model with job-splitting is presented to minimise the makespan. This technique enables the division of lots into sub-lots for operations with multiple machines available. This model, implemented in Python using the Pulp library with the CPLEX solver, performed well for small cases but failed to find solutions within acceptable time limits for medium cases.

To overcome this limitation, a hybrid genetic algorithm that integrates a local search algorithm to optimise the number and size of the sub-lots is proposed. The metaheuristic, also developed in Python, was tested on several instances and showed promising results in terms of CPU time. In evaluations with real cases from the textile industry, the CPU time was about 20 minutes for a case with 57 jobs and 35 minutes for an instance test with 67 jobs.

In addition, the lot streaming technique is explored, enabling the division of lots into sub-lots that are processed independently throughout the production flow. A mixed integer linear programming model was developed and implemented. Although the results surpassed the performance of job-splitting, the model became infeasible for certain instances due to excessive computational time.

This work highlights the importance of the techniques developed, demonstrating that metaheuristics can significantly improve the effectiveness of scheduling solutions in industry, particularly in the textile sector.

Keywords: Scheduling, Flexible job-shop, lot streaming, job-splitting, mixed-integer linear programming, hybrid genetic algorithm

Agradecimentos

A realização desta dissertação contou com vários apoios. Primeiramente, quero agradecer à minha Professora Doutora Alzira Mota, por toda a sua disponibilidade, apoio, sugestões, esclarecimentos e pela oportunidade de integrar este projeto de investigação, que se tornou uma experiência muito enriquecedora. Quero agradecer também ao meu co-orientador Professor Doutor Luís Afonso, por toda a sua disponibilidade e contribuições essenciais para a realização deste trabalho.

Gostaria de agradecer aos meus pais, irmã e família pelo apoio incondicional durante todo este percurso académico. Ao meu namorado agradeço por toda a paciência e motivação demonstrada.

Agradeço aos meus colegas académicos, pela partilha de conhecimento e espírito de companheirismo.

Por fim, agradeço a todos que de certa forma contribuíram para a realização desta dissertação.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

Lista de Acrónimos

ABC *Artificial Bee Colony*

AG Algoritmo Genético

AGH Algoritmo Genético Híbrido

AH Algoritmo Híbrido

APL Algoritmo de Pesquisa Local

B&B *Branch & Bound*

CP *Constraint Programming*

CPU *Central processing unit*

EGA *Extended Genetic Algorithm*

GGTS *Goal-Guided Tabu Search*

GRASP *Greedy Randomized Adaptive Search Procedure*

hyABC *Hybrid Artificial Bee Colony*

JS *Job-Shop*

JSF *Job-Shop* flexível

LS *Lot Streaming*

MBO *Migrating Birds Optimization*

MMBO *Modified MBO*

MRP *Manufacturing Resource Planning*

PIM Programação Inteira Mista

PL Programação Linear

PLI Programação Linear Inteira

PLIM Programação Linear Inteira Mista

PSO *Particle Swarm Optimization*

QPSO *Quantum-Behaved Particle Swarm Optimization*

SA *Simulated Annealing*

TS *Tabu Search*

VNS *Variable Neighbourhood Search*

1 Introdução

Neste capítulo estabelece-se o contexto e a relevância do problema de investigação, abordando os desafios do escalonamento em sistemas *Job-Shop* Flexíveis e a importância da divisão de lotes. Define-se a questão central da investigação e os objetivos específicos que conduzem o estudo, descreve-se a abordagem metodológica adotada para alcançar esses objetivos e apresenta-se a estrutura da tese, explicando a contribuição de cada capítulo para o desenvolvimento do tema da dissertação.

1.1 Problema de Investigação, Enquadramento e Pertinência

Nas últimas décadas, a crescente competitividade no mercado tem levado as empresas a diversificarem as suas linhas de produtos para atender às exigências variadas dos clientes. Neste cenário, a produção em sistemas *Job-Shop* tornou-se uma estratégia popular para enfrentar esses desafios, caracterizando-se pela produção de lotes de produtos que seguem rotas específicas e distintas através do processo produtivo.

A introdução da manufatura flexível em sistemas *Job-Shop* permite que várias máquinas realizem a mesma operação, o que aumenta a adaptabilidade das rotas de produção e frequentemente reduz os tempos de conclusão dos trabalhos. No entanto, essa flexibilidade torna o problema de escalonamento mais complexo do que os problemas tradicionais de *Job-Shop*.

Este estudo surge a partir de um problema de escalonamento apresentado por uma indústria têxtil que opera com um sistema de produção *Job-Shop* Flexível (JSF). A indústria enfrenta desafios significativos devido ao alto volume de produção e ao tempo limitado disponível. A literatura revela um crescente interesse em desenvolver algoritmos e estratégias inovadoras para enfrentar estes problemas complexos, com foco na melhoria da eficiência, minimização dos tempos de conclusão e otimização da utilização dos recursos.

Este trabalho tem como objetivo explorar o problema de escalonamento específico dessa empresa, utilizando duas abordagens principais para o dimensionamento dos lotes: *job-splitting* e *lot streaming*. A abordagem de *job-splitting* envolve a divisão de um lote em várias partes, ou sublotes, que são processados de forma paralela em diferentes máquinas, permitindo que múltiplas máquinas realizem a mesma operação simultaneamente. Por outro lado, a abordagem de *lot streaming* refere-se à divisão de um lote em vários sublotes que são tratados de forma independente ao longo do processo produtivo, possibilitando a execução simultânea de diferentes operações e, consequentemente, uma potencial melhoria na produtividade.

O objetivo deste estudo é desenvolver e avaliar abordagens que atendam às necessidades específicas da indústria têxtil em questão, oferecendo soluções para o problema de escalonamento. O sistema proposto atuará como uma ferramenta de apoio à decisão, que visa melhorar a eficiência do escalonamento, respondendo às necessidades da empresa.

1.2 Questões e Objetivos de Investigação

Com base no problema da indústria, o objetivo deste estudo é desenvolver e implementar um algoritmo para o problema de escalonamento em sistemas JSF, considerando o dimensionamento dos lotes. Para alcançar este objetivo geral, foram definidos os seguintes objetivos específicos:

1. **Analisar e interpretar os dados recolhidos:** Examinar os dados disponíveis para compreender melhor as características e necessidades do sistema de produção da indústria têxtil.

2. **Pesquisar as abordagens de resolução existentes:** Investigar modelos matemáticos e algoritmos aplicados ao escalonamento em sistemas *Job-Shop* Flexíveis, com foco na divisão de lotes, para entender as técnicas usadas e identificar possíveis lacunas na literatura.

3. **Propor e implementar um algoritmo:** Desenvolver um algoritmo que aborda características específicas do escalonamento JSF com divisão de lotes.

4. **Desenvolver uma ferramenta de apoio à decisão:** Criar uma ferramenta que permita a aplicação do algoritmo desenvolvido.

5. **Analisar e interpretar os resultados obtidos:** Avaliar o algoritmo implementado, interpretando os resultados.

1.3 Opções Metodológicas

Nesta seção, são apresentadas as opções metodológicas que serão adotadas para abordar o problema de escalonamento em sistemas *Job-Shop* Flexíveis com divisão de lotes. A metodologia proposta envolve uma abordagem integrada que combina análise teórica, desenvolvimento de algoritmos e avaliações práticas, conforme descrito abaixo:

1. **Revisão da Literatura:** A primeira etapa consiste em realizar uma revisão abrangente da literatura para identificar e compreender os modelos matemáticos e algoritmos existentes para o escalonamento em sistemas *Job-Shop* Flexíveis. Esta revisão foca nas técnicas matemáticas e algoritmos relevantes, com ênfase nas abordagens de divisão de lotes (*job-splitting* e *lot streaming*), visando compreender as técnicas atuais, as vantagens e limitações, e identificar lacunas na literatura.

2. **Análise e Interpretação de Dados:** A próxima etapa envolve a análise dos dados fornecidos pela indústria têxtil, incluindo informações sobre a produção, ordens de produção e recursos disponíveis. A análise desses dados irá ajudar a identificar características específicas do sistema de produção, fornecendo os requisitos necessários para o desenvolvimento do algoritmo.

3. **Desenvolvimento do Algoritmo:** Com base na revisão da literatura e na análise dos dados, será proposto um algoritmo para resolver o problema de escalonamento.

4. **Implementação e Ferramenta de Apoio à Decisão:** Será proposta uma ferramenta prática para implementar o algoritmo desenvolvido.

5. **Avaliação e Análise dos Resultados:** Após a implementação, serão realizadas análises para avaliar a eficácia do algoritmo e da ferramenta desenvolvida. Isso incluirá testes com diferentes cenários e a interpretação dos resultados obtidos.

1.4 Estrutura do Trabalho

A tese está organizada em seis capítulos, conforme descrito a seguir:

1. **Introdução:** O primeiro capítulo introduz o tema da tese, contextualizando o problema de escalonamento em sistemas *Job-Shop* Flexíveis. São apresentados o problema de investigação, o enquadramento e a pertinência do estudo, além das questões e objetivos de investigação. A secção também descreve as opções metodológicas adotadas e a estrutura geral do trabalho.

2. **Escalonamento em Sistemas *Job-Shop* Flexíveis:** O segundo capítulo explora o problema de escalonamento em sistemas *Job-Shop Flexíveis* (JSF). São discutidas as

características do problema, incluindo o dimensionamento de lotes e as técnicas de l*ot streaming* e *job-splitting*. O capítulo também aborda as medidas de desempenho relevantes e os métodos de otimização aplicáveis, incluindo métodos exatos e aproximados, bem como as principais meta-heurísticas para problemas JSF. O capítulo conclui com uma revisão dos trabalhos relacionados com o tema em estudo.

3. **Problema de Escalonamento de uma Indústria Têxtil:** No terceiro capítulo é descrito o problema da indústria têxtil, detalhando as características do sistema de produção e a descrição dos dados e recursos. O capítulo também define o desafio e os objetivos específicos relacionados com o escalonamento em contexto industrial.

4. **Modelo de Programação Linear Inteira Mista com *Job-Splitting*:** O quarto capítulo apresenta um modelo de Programação Linear Inteira Mista (PLIM) adaptado para o problema de escalonamento com *job-splitting*. São discutidos a formulação matemática do modelo, o desenvolvimento de um algoritmo genético híbrido para resolver o problema, e a criação de um sistema de apoio à decisão. Este capítulo inclui também a descrição das experiências computacionais realizadas, com a análise dos parâmetros do algoritmo genético, os resultados obtidos e o desempenho comparativo entre o modelo PLIM e o algoritmo genético híbrido.

5. **Modelo de Programação Linear Inteira Mista com *Lot Streaming*:** O quinto capítulo aborda um modelo alternativo de Programação Linear Inteira Mista que usa a técnica de *lot streaming*. O capítulo detalha a formulação matemática do modelo, o desenvolvimento de um sistema de apoio à decisão e as experiências computacionais realizadas. A análise inclui a descrição das instâncias de teste, a comparação dos tempos computacionais e a validação das soluções, bem como uma comparação entre os resultados obtidos pelos modelos PLIM utilizando as técnicas *job-splitting* e *lot streaming*.

6. **Conclusão:** O sexto e último capítulo resume as conclusões finais do trabalho e sugere possíveis direções para pesquisas futuras.

2 Escalonamento em Sistemas *Job-Shop* Flexíveis

Este capítulo faz o enquadramento do problema de escalonamento em sistemas JSF, tema central desta dissertação. Inicialmente, é feita uma contextualização e descrição do problema, seguida por uma discussão sobre o dimensionamento de lotes e a técnica de *job-splitting*. Em seguida, são apresentadas as principais medidas de desempenho utilizadas para avaliar soluções de escalonamento e os métodos de otimização usados na resolução destes problemas, incluindo métodos exatos e métodos aproximados. Por fim, são destacadas as contribuições de estudos recentes relacionados com este tipo de problemas.

2.1 O Problema de Escalonamento em Sistemas *Job-Shop* Flexíveis

O problema de escalonamento em sistemas de produção refere-se à alocação de recursos para a execução de trabalhos num determinado período, visando otimizar um ou mais objetivos, como a minimização do tempo total de produção (*makepsan*), a redução do número de trabalhos em atraso ou a maximização da utilização de recursos disponíveis.

O desenvolvimento de um escalonamento eficiente tem-se tornado numa prioridade crescente para as empresas, impulsionado pelas pressões económicas e comerciais do mercado. Nas indústrias de manufatura e serviços, o escalonamento desempenha um papel crucial na produtividade e na gestão de custos. Devido à sua importância, este processo de tomada de decisão tem atraído a atenção de pesquisadores desde 1956 (Pinedo & Chao, 1999; Shen et al., 2018; Zhang et al., 2019).

As decisões envolvidas no escalonamento devem responder a uma série de questões essenciais para garantir uma alocação eficiente dos recursos e o cumprimento dos objetivos de produção.

De acordo com Harjunkoski et al. (2014), é necessário definir quais são os trabalhos a processar, especificando as tarefas que precisam ser realizadas; onde são realizados esses trabalhos, identificando os recursos ou locais adequados para a sua execução; qual é a sequência de trabalhos, estabelecendo a ordem ideal para maximizar o fluxo produtivo; e quando são executados os trabalhos, determinando os momentos apropriados para o início e o fim de cada tarefa. A tomada de decisões eficaz em relação a estas questões é fundamental para garantir um escalonamento eficiente.

Entre os sistemas de produção, o problema de escalonamento *Job-Shop* (JS) é um sistema que surge em muitas indústrias. O escalonamento JS é um problema de otimização matemática que tem sido estudado ao longo das várias décadas devido à sua complexidade. O principal desafio está relacionado com o facto de cada trabalho possuir uma rota específica, composta por um conjunto de operações que devem ser processadas em diferentes máquinas (Dauzère-Pérès et al., 2024; Pinedo, 2012).

A solução deste problema envolve a alocação e sequenciamento dos trabalhos, considerando restrições de tempo, regras de precedência entre operações e limitações de recursos de produção. Além disso, a resolução é afetada por vários fatores, como a composição do trabalho (peça única ou lote), os tempos de *setup*, o transporte ou a data de início e de fim (Jeong et al., 1999).

Na Figura 1, pode-se visualizar a representação de um problema de escalonamento *Job-Shop*. Neste exemplo, são considerados três trabalhos distintos, cada um composto por várias operações a serem processados em três das quatro máquinas disponíveis. Para cada trabalho, há uma ordem específica de operações, e cada operação só pode ser realizada após a conclusão da anterior (Wang et al., 2021). Por exemplo, o Trabalho 2 começa na Máquina 0, segue para a Máquina 2 e é finalizado na Máquina 3.

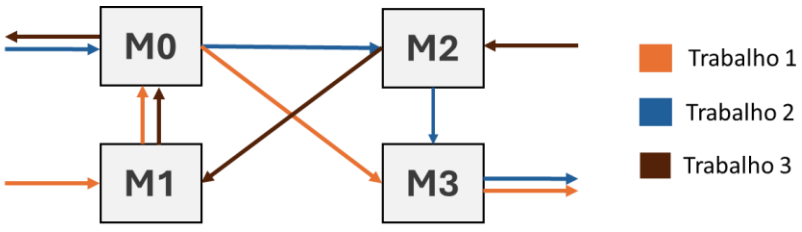


Figura 1 – Exemplo de um escalonamento *Job-Shop*

Nos sistemas de produção tradicionais, cada máquina tem capacidade para processar apenas uma única operação, sem flexibilidade de recursos, como ocorre no *Job-Shop* clássico. Contudo, com o avanço da tecnologia, surgiram máquinas de controle numérico e sistemas de produção flexíveis, permitindo que uma mesma máquina execute diferentes tipos de operações. A partir da década de 1990, essa evolução tecnológica impulsionou o estudo de uma extensão do problema de escalonamento *Job-Shop*, conhecida como *Job-Shop* Flexível (JSF), onde um

conjunto de máquinas está disponível para cada operação (Dauzère-Pérès et al., 2024; Li & Gao, 2016).

No JSF há vários centros de fabrico, cada um constituído por um conjunto de máquinas idênticas que operam em paralelo (Pinedo, 2012).

Na Figura 2, é exposto um exemplo de um problema de escalonamento JSF. Neste exemplo, são considerados 3 trabalhos e para a segunda operação existem duas máquinas capazes de a realizar. Por exemplo, o Trabalho 1 inicia na Máquina 0 e depois segue para a Máquina 1 para se realizar a segunda operação. De seguida, o trabalho é processado na Máquina 3 e o processo é finalizado.

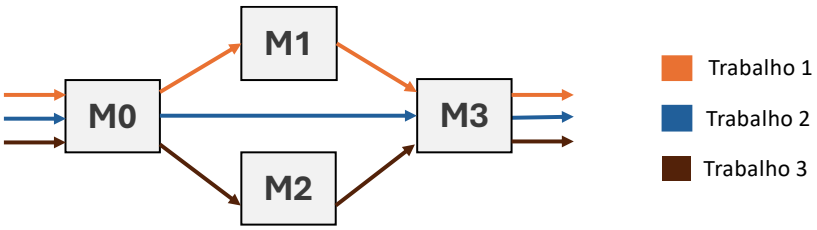


Figura 2 - Exemplo de um escalonamento *Job-Shop* Flexível

O problema de escalonamento num sistema *Job-Shop* Flexível pode ser descrito na sua forma mais elementar da seguinte forma:

- Um conjunto de trabalhos $I = \{I_1, I_2, \dots, I_n\}$, onde cada trabalho I_i consiste numa sequência ordenada de operações $O_{i1}, O_{i2}, \dots, O_{ip}$. Cada operação deve ser realizada numa ordem pré-estabelecida e não pode ser iniciada até que a anterior tenha sido concluída.

- Um conjunto de máquinas $M = \{M_1, M_2, \dots, M_m\}$, onde cada máquina M_k pode executar uma operação. No problema JSF, várias máquinas podem realizar a mesma operação, mas os tempos de processamento podem variar de acordo com a máquina selecionada. Esse tempo de processamento é representado por p_{ijk}, onde p_{ijk} é o tempo necessário para a operação O_{ij} ser processada na máquina M_k.

- O objetivo do problema é definir a sequência de processamento das operações e atribuir as operações às máquinas apropriadas, respeitando as restrições de precedência e maximizando a eficiência global do sistema, comumente representada pela minimização do *makespan*.

O problema JSF combina dois problemas interrelacionados: a atribuição de operações às máquinas e a sequenciação das operações nas máquinas, tendo em consideração as restrições

de precedência e capacidade. Esta combinação resulta num problema de otimização muito complexo, classificado como NP-difícil, tal como o clássico problema de *Job-Shop*, o que significa que encontrar soluções exatas para instâncias de grande dimensão é computacionalmente inviável (Pinedo, 2012).

2.2 Dimensionamento de Lotes

O dimensionamento de lotes é uma das estratégias usadas para melhorar a eficiência e reduzir tempos de produção em sistemas flexíveis, especialmente em ambientes JSF. Esta técnica está diretamente ligada ao *lot streaming*, que consiste em dividir lotes maiores em sublotes menores, com o objetivo de reduzir o tempo de processamento e maximizar a utilização dos recursos. Nas subseções seguintes, aprofunda-se a técnica *lot streaming* e explora-se o conceito de *job-splitting*, uma abordagem complementar que divide uma ordem de trabalho em várias partes menores, processadas em paralelo em diferentes máquinas.

2.2.1 Lot Streaming

O *lot streaming* (LS) é uma técnica avançada de escalonamento utilizada em sistemas de produção, como o *Job-Shop* Flexível. O conceito de *lot streaming* foi introduzido em 1996 por Reiter, com o objetivo de aumentar a produtividade na indústria. Desde então, muitas empresas adotaram esta técnica, por ser capaz de reduzir o tempo de produção e combater a crescente competição baseada nos tempos de entrega reduzidos (Blackburn,1991; Bockerstette & Shell, 1993; Chang & Chiu, 2005; Reiter, 1996).

De acordo com Chen e Steiner (1997), o *lot streaming* consiste em dividir um lote maior, contendo trabalhos idênticos, em vários sublotes menores que são processados de forma independente nas máquinas disponíveis. Esta divisão permite que partes de um lote comecem a ser processadas em etapas subsequentes antes que todo o lote seja concluído na etapa atual, conforme indicado por Potts & Baker (1989). Com a utilização desta técnica é possível diminuir o tempo de produção, o tempo de ciclo, o stock médio de trabalho em processamento, a quantidade de espaço de armazenamento, o *makespan* e os custos de gestão (Chen & Steiner, 1997). Além disso, o *lot streaming* possibilita a otimização do fluxo de trabalho nos sistemas de produção, ao dividir as tarefas em lotes menores, que podem ser processadas com menor inatividade entre máquinas e maior sincronização entre as operações. Este efeito pode levar a ganhos em termos de produtividade e flexibilidade na gestão de recursos e prazos (Cheng et al., 2013).

As principais características que definem o problema do LS incluem um conjunto de parâmetros e decisões que influenciam diretamente a produção. Entre eles estão:

- **Número de sublotes:** Refere-se à quantidade de sublotes em que o lote original será dividido.

- **Tamanho dos sublotes:** Determina a quantidade de itens que cada sublote contém e é um fator importante no balanceamento entre a eficiência e a complexidade operacional.

- **Configuração das máquinas:** A forma como as máquinas estão organizadas no sistema de produção afeta diretamente a sequência de operações e a eficiência do processamento dos sublotes.

- **Tempo de *setup*:** Refere-se ao tempo e aos recursos necessários para preparar as máquinas para processar cada sublote. O *setup* pode ser dependente do tamanho do sublote ou do tipo de operação, e pode influenciar consideravelmente o tempo total de produção.

- **Tempo de transferência ou remoção de sublotes**: Refere-se à duração necessária para mover ou remover sublotes entre diferentes máquinas ou etapas do processo de produção. A redução destes tempos é essencial para aumentar a eficiência do processo.

- **Função objetivo:** Por exemplo, a minimização do *makespan*.

Cheng et al. (2013) destacam que a escolha desses parâmetros depende do tipo de indústria e das especificidades do processo produtivo, e que o dimensionamento adequado dos lotes é um fator crítico para alcançar um desempenho otimizado no sistema de produção. Uma escolha inadequada do tamanho de sublotes ou do número de lotes pode gerar gargalos ou inatividade, reduzindo os ganhos de eficiência. Além disso, o tempo de *setup* e a sincronização entre diferentes operações e ordens de produção são elementos cruciais que devem ser cuidadosamente planeados.

2.2.2 Job-Splitting

O *job-splitting* é um conceito relacionado com o *lot streaming* que envolve a divisão de uma ordem de produção ou trabalho (*job*) em várias partes (ou *subjobs*), permitindo que diferentes partes sejam processadas simultaneamente em diferentes máquinas ou recursos. Enquanto o *lot streaming* foca-se na divisão de lotes (um conjunto de trabalhos idênticos) em sublotes, permitindo que diferentes sublotes avancem de maneira sobreposta no sistema, o *job-splitting* trata diretamente da divisão do lote numa operação individual. Isso significa que, em vez de cada operação ser completada numa máquina antes de passar para a próxima, o lote pode ser repartido em várias partes menores que podem ser processadas em paralelo em diferentes recursos. Esta técnica traz alguns benefícios, tais como:

- **Redução do *Makespan*:** Ao dividir o lote numa operação em várias partes e processá-las simultaneamente, contribui para a diminuição do *makespan* do sistema. Isso é particularmente útil quando o tempo de processamento em certas máquinas é longo ou a carga de trabalho é elevada.

- **Melhoria da Utilização das Máquinas:** Sublotes ou *subjobs* podem ser distribuídos para máquinas que estejam inativas, maximizando a utilização dos recursos disponíveis, balanceando melhor a carga de trabalho entre diferentes máquinas.

- **Flexibilidade no Escalonamento:** Diferentes partes de um trabalho podem ser reprogramadas para outras máquinas sem impactar significativamente o fluxo geral, permitindo ajustes mais dinâmicos para enfrentar imprevistos, como falhas de máquinas ou variações na capacidade produtiva.

- **Redução da Inatividade:** Ao invés de uma máquina ficar inativa enquanto espera que outra termine um trabalho inteiro, diferentes partes do trabalho podem ser executadas simultaneamente, mantendo a fluidez do processo produtivo, evitando tempos de espera prolongados.

Apesar das possíveis vantagens desta técnica, o *job-splitting* também apresenta desafios operacionais, tais como:

- **Sincronização:** Quando diferentes partes de um trabalho são processadas em paralelo, a sincronização entre as operações é importante. A falta de coordenação pode resultar em atrasos ou gargalos no fluxo de produção.

- ***Setup* e Custos de Transferência**: Para cada *subjob* ou sublote dividido, pode ser necessário configurar as máquinas novamente, o que pode aumentar os custos de processamento e o tempo total, especialmente se os tempos de *setup* forem elevados. Além disso, a transferência dos *subjobs* entre diferentes máquinas pode introduzir atrasos adicionais.

- **Qualidade e Consistência**: Em alguns casos, dividir um trabalho entre várias máquinas pode introduzir variações no processo produtivo, especialmente se as máquinas tiverem características de processamento ligeiramente diferentes. Sendo necessário garantir que todos os sublotes (ou *subjobs*) mantêm a qualidade e que não há discrepâncias nos resultados.

No contexto do *lot streaming*, o *job-splitting* pode ser visto como uma técnica que atua num nível mais granular. Enquanto o *lot streaming* otimiza o fluxo de lotes da gama operatória, o *job-splitting* pode ser usado para melhorar a eficiência dentro de cada operação, subdividindo os trabalhos de forma que sejam processados em paralelo. Em sistemas complexos de produção,

a utilização de uma destas técnicas ou a combinação destas técnicas pode otimizar o fluxo de produção, melhorando a produtividade e a utilização dos recursos (Tutumlu & Saraç, 2023).

2.3 Medidas de Desempenho

O problema de escalonamento visa maximizar ou minimizar uma ou mais medidas de desempenho. Historicamente, no escalonamento JSF, o objetivo principal era minimizar o *makespan*, ou seja, o tempo total necessário para completar a última tarefa de um conjunto de trabalhos. No entanto, com a evolução dos sistemas produtivos, novos critérios passaram a ser considerados, classificados em regulares e não regulares.

Os critérios regulares são funções crescentes dos tempos de conclusão dos trabalhos, o que significa que quanto mais cedo um trabalho for iniciado e concluído, melhor será o desempenho.

Os critérios não regulares, além das decisões de alocação e sequenciamento, também é considerada a decisão de temporização das tarefas. Em algumas situações, como em ambientes *Just-In-Time*, pode não ser favorável concluir um trabalho antecipadamente. Nesses casos, é preferível atrasar o início de um trabalho ou incluir tempos de espera deliberados nas máquinas. Além disso, estes critérios são relevantes em escalonamento dinâmico, onde situações em tempo real, como trabalhos urgentes, avarias de máquinas exigem ajustes constantes no planeamento (Dauzère-Pérès et al., 2024).

A Tabela 1, conforme descrito por Chaudhry e Khan (2016), apresenta as medidas de desempenho mais utilizadas no escalonamento JSF. Segundo estes autores, o *makespan* continua a ser a medida mais adotada, sendo utilizado individualmente em 44,67% dos estudos e em combinação com outras medidas em 39,59% dos casos.

Tabela 1 – Medidas de desempenho (Adaptado de Chaudhry & Khan, 2016)

Medida de Desempenho	**Notação**	**Descrição**
Makespan (tempo máximo de conclusão)	$C_{máx.}$	Tempo em que a última tarefa de uma sequência de trabalhos é finalizada.
Tempo médio de conclusão	$\bar{C}$	Tempo médio necessário para terminar o trabalho.
Tempo máximo de fluxo	$F_{máx}$	Tempo mais longo que um trabalho precisa para ser processado.
Tempo médio de fluxo	$\bar{F}$	Tempo médio que é necessário para um único trabalho ser executado.

Atraso Total	T	Diferença positiva entre o tempo de conclusão e a data de entrega de todos os trabalhos. Só existem penalidades quando os trabalhos estão atrasados.
Atraso médio	$\overline{T}$	Diferença média entre o tempo de conclusão e a data de entrega de um único trabalho.
Atraso Total ponderado	$\sum_{i=1}^{n} \alpha_i T_i$	Soma ponderada dos atrasos.
Atraso máximo	$L_{máx.}$	Atraso da tarefa com maior diferença entre o instante de conclusão e a data de entrega, sendo essa diferença positiva.
Número de trabalhos atrasados	n_T	Número de trabalhos que se encontram em atraso.
Carga Total de trabalho nas máquinas	W_T	Tempo total de trabalho utilizado pelas máquinas.
Carga de trabalho crítica da máquina	W_M	Maior carga de trabalho entre todas as máquinas.

Em sistemas onde não existe flexibilidade de máquinas, as medidas de desempenho relacionadas com a carga de trabalho perdem relevância, pois estas avaliam a alocação das operações às máquinas. No entanto, em ambientes flexíveis, tais critérios são importantes já que refletem decisões relacionadas à utilização eficiente dos recursos disponíveis. Quando se otimiza a carga de trabalho numa função objetivo, é comum combinar diferentes objetivos, pois este tipo de medida não é diretamente influenciada pelas decisões de sequenciamento ou dependentes do tempo.

Além disso, muitos problemas de escalonamento no mundo real são multiobjetivo, em que diversas medidas são consideradas simultaneamente, como a minimização do *makespan* e o atraso total. Isso reflete a complexidade dos ambientes produtivos atuais, nos quais múltiplos fatores afetam a eficiência operacional (Dauzère-Pérès et al., 2024).

2.4 Métodos de Otimização

Os problemas de escalonamento podem ser resolvidos por diversas técnicas de otimização, dependendo das suas características e complexidade. Devido à crescente exigência do mercado, as soluções precisam de ser encontradas num tempo computacional relativamente curto.

As abordagens para resolver problemas de escalonamento, em particular, o problema JSF podem ser divididas em três categorias principais: métodos exatos, heurísticas e meta-heurísticas. Na Figura 3 é possível visualizar de forma esquemática os diferentes métodos de otimização.

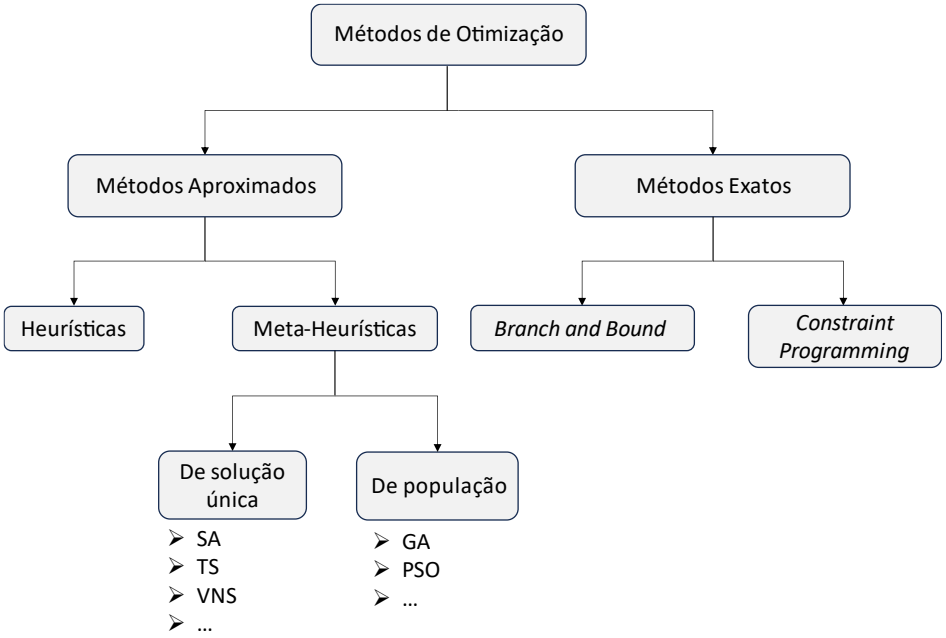


Figura 3 – Métodos de Otimização

2.4.1 Métodos Exatos

Os métodos exatos garantem soluções ótimas, o que pode implicar um elevado tempo computacional, pelo que normalmente são utilizados em problemas de pequena dimensão (Muthuraman & Venkatesan, 2017). As abordagens mais comuns incluem os algoritmos *Branch and Bound* (B&B) e os modelos de Programação Linear Inteira Mista (PLIM). Como alternativa promissora, a *Constraint Programming* (CP) tem atraído a atenção dos investigadores, mostrando um desempenho superior à PLIM em várias instâncias, especialmente em problemas de grande dimensão (Dauzère-Pérès et al., 2024; Naderi et al., 2023).

O *Branch and Bound* foi proposto por Land e Doig (Land & Doig, 1960). Este algoritmo explora todas as soluções possíveis de um problema, organizando-as numa estrutura de árvore e armazenando soluções parciais, chamadas de subproblemas. O processo começa decompondo o problema inicial, que é representado pelo nó raiz da árvore, e prossegue construindo uma árvore de pesquisa, onde os nós folha correspondem às soluções finais e os nós internos aos subproblemas derivados. A construção da árvore é realizada por quatro operadores:

ramificação, delimitação, poda e seleção. A ramificação divide o problema em subproblemas menores, enquanto o operador de delimitação calcula limites inferiores do custo ideal para cada subproblema. A poda elimina subproblemas cujo limite inferior exceda o custo da melhor solução já encontrada. Por fim, o operador de seleção escolhe o próximo subproblema a ser explorado, de acordo com a estratégia de pesquisa adotada. No final, o algoritmo retorna a melhor solução encontrada após a exploração completa da árvore (Gmys et al, 2020; Morrison et al., 2016).

Os modelos de Programação Inteira Mista (PIM), por sua vez, são populares tanto na indústria quanto na literatura, com três formulações principais: a disjuntiva, a baseada em classificação e a indexada ao tempo. Além dessas, há outros modelos que correspondem a combinações ou variações destas três formulações (Ku & Beck, 2016).

O *Constraint Programming* (CP) é uma alternativa à PIM, modelando problemas como um conjunto de restrições a serem satisfeitas. CP é particularmente útil em ambientes onde o problema pode ser descrito através de variáveis de intervalo e restrições globais (Zhang et al., 2023).

2.4.2 Métodos Aproximados

Os métodos aproximados incluem heurísticas e meta-heurísticas. Embora não garantam soluções ótimas, são mais eficientes em termos de tempo computacional, sendo aplicados em problemas de maior dimensão e mais complexos.

As heurísticas são regras práticas usadas para gerar soluções ou explorar vizinhanças de soluções candidatas (pesquisa local). Existem heurísticas para a alocação de operações às máquinas, como a do tempo mínimo de processamento ou de conclusão, bem como regras para sequenciamento de operações. Embora simples, são úteis e podem ser combinadas com outros algoritmos (Gao et al., 2016; Pezzella et al., 2008).

As meta-heurísticas são estratégias de pesquisa mais avançadas que visam melhorar a eficiência na resolução de problemas complexos. Elas podem ser divididas em dois tipos principais: de solução única e de população.

Nas meta-heurísticas de solução única, é desenvolvida uma trajetória de pesquisa através de abordagens de pesquisa local que exploram a vizinhança da solução atual para melhorá-la, focando-se em soluções individuais. Exemplos deste tipo de meta-heurísticas incluem o *Simulated Annealing* (SA), o *Tabu Search* (TS) e o *Variable Neighbourhood Search* (VNS), todas utilizadas na resolução de problemas de JSF (Dauzère-Pérès et al., 2024; Muthuraman & Venkatesan, 2017).

Por outro lado, as meta-heurísticas baseadas em população lidam com um conjunto de soluções simultaneamente. De acordo com Dauzère-Pérès et al. (2024), o Algoritmo Genético (AG) é o

método mais usado para problemas JSF, embora o *Particle Swarm Optimization* (PSO) também seja bastante popular.

A combinação de diferentes técnicas, conhecida como algoritmos híbridos, tem se mostrado eficaz para resolver problemas complexos de grande dimensão. Ao integrar métodos exatos, heurísticas e meta-heurísticas, pretende-se aproveitar os pontos fortes de cada abordagem para obter soluções mais robustas (Dauzère-Pérès et al., 2024; Muthuraman & Venkatesan, 2017).

2.4.3 Principais Meta-Heurísticas para Problemas *Job-Shop* Flexível

As meta-heurísticas são muito utilizadas para resolver problemas de JSF. As comumente usadas são:

- ***Simulated Annealing* (SA):** Introduzido por Kirkpatrick e Vecchi em 1983, o SA é inspirado no processo de arrefecimento metálico. Este método explora temporariamente soluções de qualidade inferior para escapar de mínimos locais, com uma taxa de aceitação controlada por um parâmetro de temperatura que diminui progressivamente ao longo da procura (Kirkpatrick et al., 1983; Kuruoglu et al., 2023).

- ***Tabu Search* (TS):** Proposto por Glover, em 1986, o TS utiliza uma estrutura de memória, conhecida como "lista tabu", para registar movimentos anteriores e impedir a repetição de soluções. Isso possibilita a exploração de novas áreas no espaço de soluções, evitando a estagnação em ótimos locais (Dueck & Scheuer, 1990; Glover, 1986).

- ***Variable Neighbourhood Search* (VNS):** Introduzido por Mladenović e Hansen, em 1997. O VNS alterna entre diferentes vizinhanças de soluções para evitar ótimos locais. Este método tem apresentando bom desempenho tanto em problemas de otimização discretos quanto contínuos (Chaudhry & Khan, 2016; Mladenović & Hansen, 1997).

- **Algoritmo Genético (AG):** Proposto por John Henry Holland, em 1975. O AG é inspirado em processos evolutivos. Este método trabalha com uma população de soluções e utiliza operadores como cruzamento e mutação para gerar novas soluções, permitindo a diversidade e melhorando a qualidade geral da população (Cavallaro et al., 2024; Holland, 1992).

- ***Particle Swarm Optimization* (PSO):** Desenvolvido por Eberhart e Kennedy, em 1995. O PSO é inspirado no comportamento coletivo de enxames. O PSO ajusta as soluções individuais com base nas melhores soluções do grupo, convergindo para uma solução ótima (Kennedy & Eberhart, 1995; Shojaee et al., 2024).

2.5 Trabalhos Relacionados

Nesta seção, são apresentados diversos estudos sobre o problema de *Job-Shop* Flexível, destacando as particularidades de cada trabalho, os métodos utilizados na resolução do problema, bem como as principais conclusões e lacunas identificadas.

Singh e Mahapatra (2016) abordaram o problema JSF utilizando o *Quantum-Behaved Particle Swarm Optimization* (QPSO). O QPSO foi utilizado para mitigar a tendência do PSO de ficar preso em ótimos locais. Para evitar a convergência prematura, foi incorporada a mutação, técnica usada nos algoritmos genéticos. Além disso, os autores usaram sequências caóticas em vez de sequências aleatórias, como forma de diversificar a população e assim diminuir a possibilidade de convergência em mínimos locais. Especificamente, foi adotada a sequência caótica de séries temporais do mapa logístico e os resultados mostraram que o QPSO gerou soluções melhores ou equivalentes às encontradas na literatura. No entanto, o tempo computacional poderia ser reduzido com a hibridização do QPSO com técnicas de pesquisa local.

Li e Gao (2016) propuseram um Algoritmo Híbrido (AH), combinando Algoritmos Genéticos (AG) e *Tabu Search*, com o objetivo de minimizar o *makespan*. A combinação dessas técnicas permitiu um equilíbrio eficaz entre a intensificação e a diversificação da pesquisa, aproveitando a capacidade de exploração global do AG e a eficiência da pesquisa local do TS. O algoritmo também integra operadores genéticos, estrutura de vizinhança e métodos eficientes de codificação. O AH mostrou ser eficiente na obtenção de novas soluções em problemas JSF, independentemente do tempo computacional ou da precisão das soluções. Futuras melhorias poderiam envolver a integração do método com técnicas multiobjectivo, dado que o algoritmo proposto é eficaz apenas para problemas uni objetivo.

Zhang e Yang (2016) trataram do problema JSF num contexto real de uma fábrica aeronáutica, com características específicas como dias de trabalho flexíveis e a preempção de operações. Três objetivos foram considerados: minimizar atrasos nos pedidos, o tempo de inatividade e as horas extras das máquinas. Três métodos foram propostos: um heurístico baseado em regras de prioridade, o *Goal-Guided Tabu Search* (GGTS) e o *Extended Genetic Algorithm* (EGA). Entre eles, o GGTS utilizando uma função de vizinhança específica mostrou-se mais eficiente e rápido. Além disso, o GGTS mostrou-se competitivo com o EGA, demonstrando aplicabilidade em situações reais e potencial para ser aprimorado com estudos sobre combinações de movimentos e avaliações de movimentos aproximados.

Shen et al. (2018) focaram o JSF com tempos de *setup* dependentes da sequência, visando minimizar o *makespan*. Um modelo matemático foi apresentado e um algoritmo de TS com funções de vizinhança inovadoras e uma estrutura de diversificação específica. O algoritmo TS demonstrou melhor desempenho em comparação com o modelo matemático e outras meta-heurísticas em instâncias pequenas. Futuras extensões poderiam considerar múltiplos recursos e outros objetivos além do *makespan*. Além disso, a modelação de recursos necessários para realizar os *setups* e que não podem ser utilizados em simultâneo.

Defersha e Bayat (2018) desenvolveram um algoritmo genético assistido por programação linear para resolver um problema de escalonamento JSF com *lot streaming.* Foram testadas duas abordagens para resolver subproblemas de Programação Linear (PL), uma direta e outra otimizada para reduzir a carga computacional. Concluiu-se que a hibridização da PL com o AG permitiu resolver eficientemente problemas de JSF com *lot streaming*.

Bożek e Werner (2018) propuseram um método de otimização em duas etapas para o JSF com *lot streaming* e otimização do tamanho de sublote, aplicado numa fábrica de fixadores. A primeira etapa minimizou o *makespan*, enquanto a segunda maximizou os tamanhos dos sublotes sem aumentar o *makespan*. O estudo concluiu que algoritmos metaheurísticos, como *Tabu Search*, superam métodos baseados em PLIM para instâncias grandes, sugerindo a inclusão de novos objetivos, como o tempo médio de fluxo e atraso.

Tutumlu e Saraç (2023) resolveram um problema JSF com *job-splitting*, utilizando um modelo de PLIM e um Algoritmo Genético Híbrido (AGH). No modelo proposto o tamanho e o número de sublotes não são predefinidos ou limitados, sendo o principal objetivo minimizar o *makespan*. No entanto, o modelo PLIM não conseguiu gerar soluções viáveis para problemas de grande dimensão. Em contrapartida, o AGH, que incorpora um Algoritmo de Pesquisa Local (APL) para determinar de forma eficiente o tamanho dos sublotes, mostrou-se eficaz, resultando numa redução significativa no *makespan*.

Meng et al. (2018) abordaram o problema de JSF com sobreposição de operações, uma técnica que permite que a operação subsequente inicie assim que parte da operação atual seja concluída, desde que a máquina esteja disponível (Demir & İşleyen, 2014). Esta técnica permite a execução simultânea de operações sucessivas no mesmo trabalho, aumentando a flexibilidade e a produtividade do sistema. Para resolver o problema, foi proposto o algoritmo *Hybrid Artificial Bee Colony* (hyABC), cujo objetivo é minimizar o tempo total de fluxo. A metodologia hyABC combina as vantagens do *Artificial Bee Colony* (ABC) com a meta-heurística *Migrating Birds Optimisation* (MBO). Devido à baixa capacidade de exploração do ABC, foi integrada uma versão modificada do MBO (MMBO) para equilibrar a pesquisa global e local. O algoritmo proposto superou cinco métodos de última geração, tanto em termos de qualidade das soluções quanto de convergência. Melhorias futuras poderiam considerar aspetos reais, como a inclusão de *buffers* limitados e prazos de entrega.

Boyer et al. (2021) introduziram um problema JSF com base numa situação real numa fábrica de anéis laminados, incorporando restrições como capacidade da máquina, tempos de *setup* dependentes da sequência e tempos de espera. Utilizaram um modelo PLIM e um modelo de Programação por Restrições (CP), além de uma meta-heurística baseada em *Greedy Randomized Adaptive Search Procedure* (GRASP), que se mostrou mais eficaz para instâncias grandes.

Defersha et al. (2022) investigaram o problema JSF com restrições de operadores de *setup*. No modelo proposto é considerado o tempo de *setup* dependente da sequência e o equilíbrio da carga de trabalho entre os operadores de *setup*. Os autores desenvolveram um algoritmo de *Simulated Annealing* com os seguintes passos: representação da solução, inicialização, função avaliação, operadores de movimento, critério de aceitação/ rejeição, cronograma de arrefecimento e critérios de paragem. O algoritmo mostrou-se competitivo em termos de *makespan*, comparando-se favoravelmente com métodos paralelos. Limitações futuras incluem a aplicação a estudos de casos industriais reais.

Park e Ham (2022) abordaram o problema JSF com foco na otimização do consumo de energia. Os autores propuseram modelos de Programação Linear Inteira (PLI) e CP, em que o primeiro objetivo é minimizar o *makespan* e o segundo é minimizar o custo total de energia. Através das experiências computacionais, verifica-se que o CP supera o modelo PLI. Além disso, os modelos desenvolvidos demonstram que é possível economizar energia, em média, em 6,9% sem comprometer a produtividade.

Na Tabela 2, encontra-se um resumo das técnicas de otimização utilizadas em cada um dos trabalhos expostos anteriormente.

Tabela 2 – Resumo dos trabalhos relacionados

Tipo de problema	**Método de Otimização**	**Autores**
JSF	QPSO	Singh e Mahapatra (2016)
JSF	Algoritmo Híbrido (AG + TS)	Li e Gao (2016)
JSF com dias de trabalho flexíveis, preempção e sobreposição de operações	Método heurístico baseado em regras de prioridade; GGTS; EGA	Zhang, J., & Yang, J. (2016)
JSF com tempos de *setup* dependentes da sequência	TS	Shen et al. (2018)
JSF com *lot streaming*	Algoritmo Genético assistido por programação linear	Defersha e Bayat (2018)
JSF com l*ot streaming*	Escalonamento (CP; TS) *Lot streaming* (PLIM; *Greedy Constructive Algorithm*)	Bożek e Werner (2018)
JSF com *job-splitting*	PLIM + AGH	Tutumlu e Saraç (2023)

JSF com sobreposição de operações	hyABC (ABC + MMBO)	Meng et al. (2018)
JSF com restrições de capacidade da máquina, tempos de espera, tempos de *setup* dependentes da sequência e tempos de atraso	PLIM; CP; GRASP	Boyer et al. (2021)
JSF com restrição de operadores de *setup*	SA	Defersha et al. (2022)
JSF sob preços de tempo de uso e tempo de inatividade programada	PLI; CP	Park e Ham (2022)

3 Problema de Escalonamento de uma Indústria Têxtil

Este capítulo descreve o problema real de escalonamento de uma indústria têxtil especializada na produção de cachecóis e gorros personalizados para eventos desportivos. Este mercado exige tempos de resposta rápidos e elevada taxa de pontualidade na entrega dos produtos, uma vez que os prazos acordados com os clientes são rigorosos e as penalidades por atrasos podem resultar em prejuízos significativos. A empresa enfrenta um problema complexo ao tentar equilibrar a procura constante por produtos personalizados com as restrições operacionais e de capacidade. Diariamente, lida com o desafio de otimizar o seu processo de produção, considerando a variabilidade nas ordens de fabrico e as limitações dos recursos disponíveis.

3.1 Características do Sistema de Produção

O sistema de produção da empresa categoriza os cachecóis e gorros em diferentes famílias de produtos. Cada família inclui vários modelos, e cada modelo possui uma sequência específica de operações e tempos de fabrico distintos. Os recursos necessários para cada operação são alocados de acordo com a natureza do produto e a especificidade das operações**.**

Os postos de trabalho na empresa estão equipados com máquinas que podem produzir exclusivamente cachecóis, exclusivamente gorros ou ambos os tipos de produtos. As máquinas são distribuídas em diferentes postos de trabalho, e as operações de produção devem ser alocadas de acordo com as capacidades e a disponibilidade das máquinas.

De acordo com as características do sistema, este é classificado como um sistema *Job-Shop* Flexível.

3.2 Descrição dos dados e recursos

As ordens de fabrico são geradas por um sistema *Manufacturing Resource Planning* (MRP) com base nos pedidos dos clientes, que têm datas de entrega específicas e a maioria das vezes urgentes. A empresa opera 5 dias por semana, com máquinas a funcionar continuamente 24 horas por dia, em três turnos de 8 horas cada. Além disso, existem tarefas de mão de obra que funcionam durante 7,5 horas por dia.

Os dados das ordens de fabrico estão organizados num ficheiro Excel, contendo informações detalhadas na folha "Instância". A Tabela 3 apresenta um exemplo das informações disponíveis.

Tabela 3 - Folha "Instância"

ID	OF	Pre	Mod	Op	Data de Entrega	Hora	Qt	Fam	Posto
1	202401		J2	K	10/10/2024	17:30	500	FP1	KN
2	202401	1 ID	J2	M	10/10/2024	17:30	500	FP1	MC

A coluna "ID" identifica a linha de dados, a "OF" refere-se à ordem de fabrico, e a "Pre" mostra as operações precedentes, se existirem. A coluna "Mod" indica o modelo do produto a ser produzido e "Op" a operação específica. As colunas "Data de entrega" e "Hora" apresentam as informações de entrega da encomenda, enquanto "Qt" mostra a quantidade total da ordem a ser produzida. A coluna "Fam" define a família de produtos e a coluna "Posto" indica o posto de trabalho onde a operação pode ser realizada.

Na folha "Recursos", disponível na Tabela 4, encontram-se detalhes sobre a disponibilidade dos postos de trabalho, máquinas, tempos de *setup*, e tempos de produção. Esta folha fornece uma visão detalhada sobre a capacidade de produção e as restrições de cada posto de trabalho.

Tabela 4 – Folha "Recursos"

Disp (min)	Posto de Trabalho	Máquinas	*Setup* (min)	Tempo por repetição	Nº peças por repetição	Peças/Min (Cachecóis)
1440	KN	KN0	5	16	8	0,5
1440	KN	KN1	5	16	8	0,5
450	MC	MC0	-	-	-	-

Os dados mostram a disponibilidade diária das máquinas (coluna "Disp (min)"), o tempo necessário para o *setup*, o tempo de produção por repetição, o número de peças produzidas por repetição e a taxa de produção por minuto para cachecóis. Observa-se que algumas máquinas não produzem certos produtos, como a máquina MC0, que não produz cachecóis. Porém, existem outras que podem produzir ambos os tipos de produtos.

3.3 Desafio e Objetivos

O principal desafio da empresa é garantir que todas as ordens sejam concluídas dentro dos prazos acordados, visto que as penalidades por atrasos podem ter um impacto significativo nos negócios. Assim, a empresa visa maximizar a utilização dos recursos disponíveis e minimizar o tempo de produção para garantir o cumprimento rigoroso dos prazos. Para lidar com esta complexidade, é crucial ter um sistema que forneça soluções eficazes que considerem tanto a disponibilidade das máquinas quanto as especificidades das ordens de fabrico.

4 Modelo de Programação Linear Inteira Mista com *Job-Splitting*

Este capítulo apresenta a formulação matemática e a resolução do problema de escalonamento com *job-splitting*. Primeiramente, um modelo programação linear inteira mista (PLIM1) é detalhado, com o objetivo de minimizar o *makespan*. Em seguida, é introduzido um algoritmo genético híbrido (AGH), que integra técnicas evolutivas e um algoritmo de pesquisa local para melhorar o desempenho computacional e a qualidade das soluções encontradas. A estrutura do algoritmo é apresentada de forma a abranger desde a representação das soluções até os operadores genéticos e o critério de paragem. Para avaliar a eficácia do AGH, são apresentadas experiências computacionais, onde são determinados os parâmetros do algoritmo, testados diferentes cenários e analisados os tempos computacionais. Por fim, é feita uma comparação entre o desempenho do AGH e o modelo PLIM1, em que é avaliada a eficiência de ambos em termos de tempos de execução e qualidade das soluções obtidas.

4.1 Formulação Matemática

Com base no problema descrito no Capítulo 3, é apresentada a formulação matemática de um modelo de otimização. A abordagem adotada utiliza a técnica *job-splitting*, permitindo que um trabalho seja dividido em lotes menores e processado por diferentes máquinas, com o objetivo principal de aumentar a flexibilidade no uso dos recursos, otimizar a taxa de utilização e reduzir os tempos de inatividade das máquinas.

O modelo de otimização proposto para resolver este problema é baseado no trabalho de Tutumlu e Saraç (2023). Este modelo é desenvolvido com base nos seguintes pressupostos:

- Cada máquina pode processar apenas um trabalho de cada vez;

- Os tempos de processamento e de *setup* são determinísticos, o que significa que são conhecidos e não variam ao longo do tempo;

- Para operações em que há mais de uma máquina disponível, o trabalho pode ser dividido entre as diferentes máquinas, podendo os tamanhos dos lotes ser distintos;

- Cada operação requer um tempo de *setup*, sendo este diferente para cada máquina envolvida no processo;

- No caso de haver divisão de um trabalho entre várias máquinas, considera-se um tempo de *setup* individual para cada máquina utilizada;

- As rotas de produção de cada trabalho são fornecidas previamente;

- Cada máquina tem um número mínimo de peças que deve produzir por repetição;

- Cada máquina realiza apenas um tipo específico de operação.

Nas Tabelas 5, 6, 7 e 8 são apresentados os conjuntos, índices, parâmetros e variáveis de decisão que compõem o modelo de Programação Linear Inteira Mista (PLIM1).

Tabela 5 – Conjuntos do PLIM1

Conjuntos	**Descrição**
$\boldsymbol{I = \{1, \dots, n\}}$	Conjunto de trabalhos
$\boldsymbol{J = \{1, \dots, t\}}$	Conjunto de operações
$\boldsymbol{M = \{1, \dots, m\}}$	Conjunto de máquinas
$\boldsymbol{N = \{N_1, \dots, N_n\}}$	Conjunto com o número de operações por trabalho, onde N_i representa o número de total de operações do trabalho i, com , $i = 1, \dots, n$
$\boldsymbol{E = \{E_1, \dots, E_n\}}$	Conjunto com as quantidades de produção, onde E_i indica o volume da encomenda do trabalho i, com $i = 1, \dots, n$
$\boldsymbol{L_k = \{1, \dots, \delta\}}$	L_k é o conjunto com a sequência de operações da máquina k, $k = 1, \dots, m$, onde $\delta = \max_k \sum_{i \in I} \sum_{j \in N_i} u_{i,j,k}$
$\boldsymbol{M_j = \{\eta_1, \dots, \eta_{O_j}\}}$	M_j é o conjunto representativo das máquinas do centro de trabalho que realizam a operação j, com $j = 1, \dots, t$, sendo O_j o número total de máquinas disponíveis para realizar a operação j.

Tabela 6 – Índices do PLIM1

Indices	Descrição
$\boldsymbol{i}, \boldsymbol{i'} \in \boldsymbol{I}$	Indicam um trabalho
$\boldsymbol{j}, \boldsymbol{j'} \in \boldsymbol{N_i}$	Indicam uma operação do trabalho i
$\boldsymbol{k}, \boldsymbol{k'} \in \boldsymbol{M_j}$	Indicam uma máquina de um centro de trabalho que realiza a operação j
$\boldsymbol{l}, \boldsymbol{l'} \in \boldsymbol{L_k}$	Indicam uma posição de um trabalho na sequência das operações da máquina k

Tabela 7 – Parâmetros do PLIM1

Parâmetros	Descrição
$\boldsymbol{P_k}$	Tempo de processamento da máquina k, com $k = 1, \dots, m$
$\boldsymbol{S_k}$	Tempo de *setup* da máquina k, com $k = 1, \dots, m$
$\boldsymbol{u_{i,j,k}}$	1, se a máquina k é capaz de processar a operação j do trabalho i
	0, caso contrário
$\boldsymbol{h_i}$	Máximo do número mínimo de peças a produzir por repetição entre todas as máquinas dos centros de trabalho onde são realizadas as operações j do trabalho i, correspondendo ao tamanho mínimo de lote do trabalho i.
$\boldsymbol{\Omega}$	Número positivo muito grande

Tabela 8 – Variáveis de decisão do PLIM1

Variáveis de decisão	Descrição
$\boldsymbol{C_{i,j,k}}$	Tempo de conclusão do trabalho i na operação j na máquina k
$\boldsymbol{C_{max}}$	Tempo de conclusão do último trabalho (*makespan*)
$\boldsymbol{\sigma_{i,j,k}}$	Tamanho do sublote do trabalho i na operação j na máquina k
$\boldsymbol{x_{i,j,k,l}}$	1, se o trabalho i na operação j é processado na máquina k na posição l
	0, caso contrário

A formulação matemática do problema JSF é baseada no modelo proposto por Tutumlu e Saraç (2023) e é descrita a seguir.

$$\text{minimizar } f = C_{max} \tag{1}$$

Sujeito a:

$$\sum_{k \in M_j} \sigma_{i,j,k} = E_i \quad \forall_{i,j \leq N_i} \tag{2}$$

$$\sigma_{i,j,k} \geq h_i x_{i,j,k,l} \quad \forall_{i,j,k,l,j \leq N_i, k \epsilon M_j} \tag{3}$$

$$\sigma_{i,j,k} \leq E_i \sum_{l \in L_k} x_{i,j,k,l} \quad \forall_{i,j,k,l,j \leq N_i, k \epsilon M_j} \tag{4}$$

$$x_{i,j,k,l} \leq u_{i,j,k} \quad \forall_{i,j,k,l,j \leq N_i, k \epsilon M_j} \tag{5}$$

$$C_{i,j,k} + \Omega * (1 - x_{i,j,k,l}) \geq \sigma_{i,j,k} P_k + S_k \quad \forall_{i,j,k,l,j=1,l=1,k \epsilon M_j} \tag{6}$$

$$\begin{gathered} C_{i,j,k} + \Omega * \left(2 - x_{i,j,k,l} - x_{i',j',k,l-1}\right) \\ \geq C_{i\prime,j\prime,k} + \sigma_{i,j,k} P_k + S_k \ \forall_{i,j,i',j',k,j=1,l>1,i \neq i\prime,k \epsilon M_j} \end{gathered} \tag{7}$$

$$\begin{gathered} C_{i,j,k} + \Omega * \left(2 - x_{i,j,k,l} - x_{i,(j-1),k',l'}\right) \geq C_{i,(j-1),k'} + \sigma_{i,j,k} P_k + S_k \\ \forall_{i,j,k,l,k',l',j>1,l=1,j \leq N_i, k \epsilon M_j, k\prime \epsilon M_{j-1}} \end{gathered} \tag{8}$$

$$\begin{gathered} C_{i,j,k} + \Omega * \left(2 - x_{i,j,k,l} - x_{i,(j-1),k',l'}\right) \geq C_{i,(j-1),k'} + \sigma_{i,j,k} P_k + S_k \\ \forall_{i,j,k,l,k',l',j>1,l>1,j \leq N_i, k \epsilon M_j, k\prime \epsilon M_{j-1}} \end{gathered} \tag{9}$$

$$\begin{gathered} C_{i,j,k} + \Omega * \left(2 - x_{i,j,k,l} - x_{i',j',k,l-1}\right) \\ \geq C_{i\prime,j\prime,k} + \sigma_{i,j,k} P_k + S_k \quad \forall_{i,j,k,l,i',j',l>1,j \leq N_i, k \epsilon M_j, i \neq i'} \end{gathered} \tag{10}$$

$$l - l' \geq 1 - \Omega * \left(2 - x_{i,j,k,l} - x_{i,j',k,l'}\right) \quad \forall_{i,j,k,l,j',l',j \leq N_i, k \epsilon M_j, j \neq j', l \neq l'} \tag{11}$$

$$\sum_{i \in I} \sum_{j \in N_i} x_{i,j,k,l} - \sum_{i' \in I} \sum_{j' \in N_{i'}} x_{i',j',k,l-1} \leq 0 \ \forall_{k,l,l>1,k \epsilon M_j} \tag{12}$$

$$\sum_{l \in L_k} x_{i,j,k,l} \leq 1 \quad \forall_{i,j,k,j \leq N_i, k \epsilon M_j} \tag{13}$$

$$\sum_{i \in I} \sum_{j \in N_i} x_{i,j,k,l} \leq 1 \ \forall_{k,l,k \epsilon M_j} \tag{14}$$

$$C_{max} \geq C_{i,j,k} \quad \forall_{i,j,k,j \leq N_i, k \epsilon M_j} \tag{15}$$

$$C_{i,j,k} \geq 0 \quad \forall_{i,j,k} \tag{16}$$

$$C_{max} \geq 0 \tag{17}$$

$$x_{i,j,k,l} \in \{0,1\} \quad \forall_{i,j,k,l} \tag{18}$$

$$\sigma_{i,j,k} \geq 0 \text{ e inteiro} \quad \forall_{i,j,k,l,j \leq N_i, k \epsilon M_j} \tag{19}$$

A equação (1) define a função objetivo, em que o problema consiste na minimização do *makespan*. A equação (2) garante que o trabalho i na operação j é atribuído a máquinas e que a soma dos tamanhos dos sublotes nessa operação é igual à quantidade total a produzir para o respetivo trabalho. A equação (3) permite que o tamanho dos sublotes seja maior ou igual a h_i. Além disso, a restrição (4) garante que o tamanho do sublote é zero se não for atribuído o trabalho i na operação j à máquina k em análise. A equação (5) garante que as operações são atribuídas às máquinas capazes de as realizar. As equações (6) e (7) determinam o tempo de conclusão de uma operação quando esta corresponde à primeira operação de um trabalho. No caso da equação (6), a operação em estudo é a primeira a ser realizada na máquina atribuída. Por outro lado, na equação (7) a operação não é a primeira e é considerado o tempo de conclusão da operação anterior na mesma máquina. As equações (8) e (9) correspondem à situação em que a operação em análise não corresponde à primeira operação de um trabalho. No caso da equação (8), a operação é a primeira a ser realizada na máquina atribuída, pelo que é considerado o tempo de conclusão da operação anterior do mesmo trabalho realizada noutra máquina. Na equação (9), a posição da operação na máquina não é a primeira, pelo que é considerado o tempo de conclusão da operação anterior do mesmo trabalho. Na equação (10), garante-se a determinação do tempo de conclusão para as operações que não se encontram na primeira posição da máquina e por isso considera-se o tempo de conclusão da operação anterior na mesma máquina. As equações (11) e (12) garantem que as operações de um trabalho são realizadas de forma sequencial. Relativamente à equação (13), esta garante que, no máximo, cada operação de cada trabalho é atribuída a uma máquina numa dada posição. Na equação (14) é estabelecido que no máximo cada operação é atribuída a cada sequência de cada máquina. A equação (15) corresponde ao cálculo do *makespan*. As restrições (16),(17) e (19) são restrições de sinal e, por fim, a restrição (18) impõe que as variáveis de decisão $x_{i,j,k,l}$ sejam do tipo binário.

4.2 Algoritmo Genético Híbrido

Embora o modelo de Programação Linear Inteira Mista tenha sido formulado para resolver o problema, os *solvers* tradicionais não conseguem obter soluções num tempo útil devido à complexidade e dimensão do problema. Para contornar esta limitação, propõe-se a utilização de um AGH inspirado no trabalho de Tutumlu e Saraç (2023).

Os algoritmos genéticos são técnicas inspiradas nos princípios Darwinianos da evolução das espécies e da genética, conforme descrito por Goldberg (1989). São algoritmos probabilísticos que fornecem um mecanismo de pesquisa paralela e adaptativa baseado no princípio de sobrevivência dos mais aptos e na reprodução. Estes algoritmos trabalham com uma população de possíveis soluções e utilizam operações de seleção, cruzamento e mutação para evoluir ao longo de várias gerações, na procura por soluções progressivamente melhores. No operador de seleção, os indivíduos mais aptos são escolhidos para gerar descendentes. Durante o cruzamento, o material genético de dois indivíduos é combinado para criar novos indivíduos, enquanto a mutação introduz variações aleatórias, aumentando a diversidade da população e evitando a convergência prematura para ótimos locais.

Neste estudo, o AGH proposto inclui um componente de pesquisa local, que visa melhorar a qualidade das soluções, gerando novos indivíduos de elite. O fluxograma do AGH proposto pode ser visualizado na Figura 4, enquanto os detalhes completos do algoritmo serão descritos nas subseções seguintes.

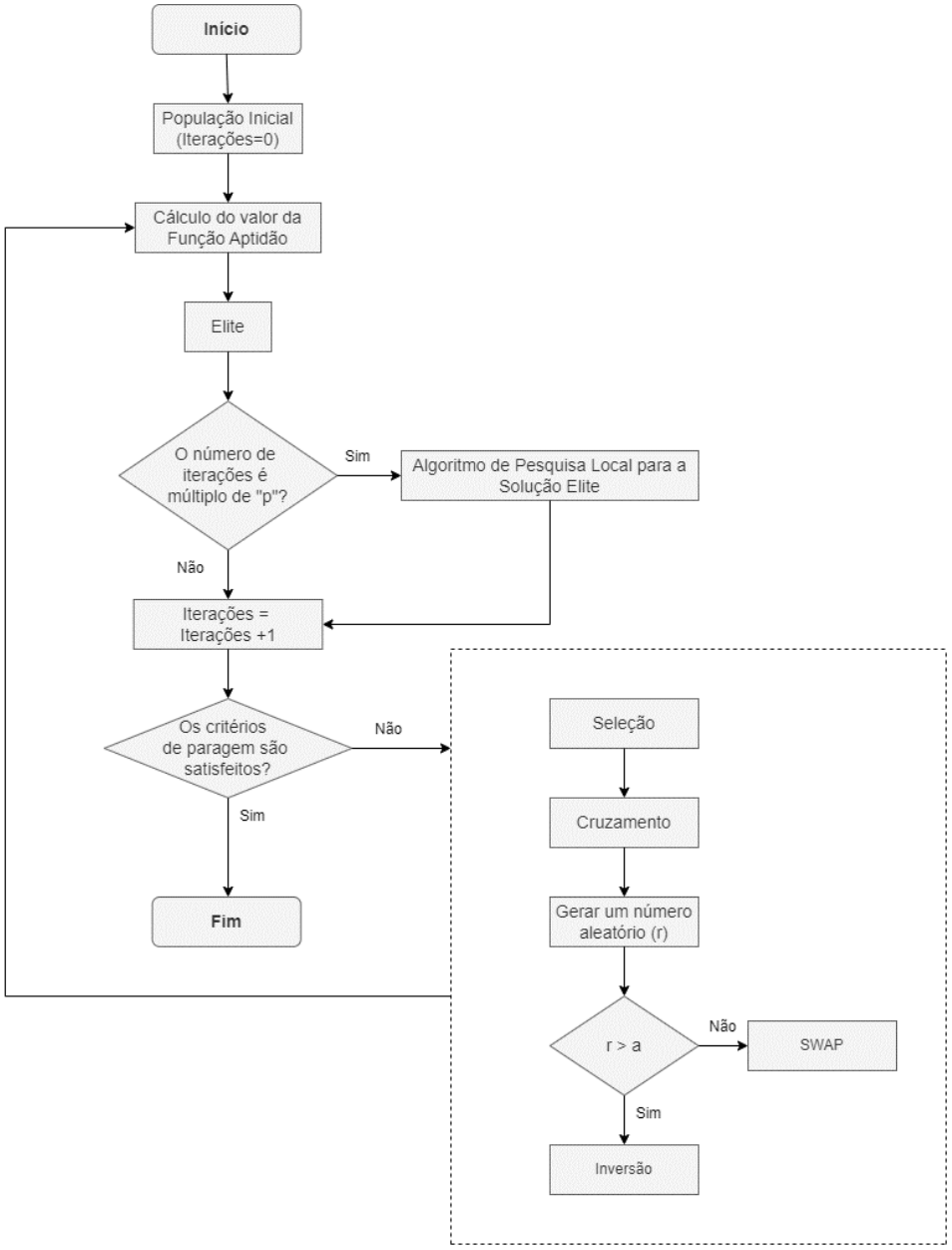


Figura 4 – Fluxograma do AGH (adaptado de Tutumlu e Saraç, 2023)

4.2.1 Representação da Solução

A solução (ou indivíduo) é representada por duas matrizes. A primeira matriz indica os tamanhos dos sublotes atribuídos a cada máquina para cada operação. Já a segunda matriz especifica em que máquina cada operação é realizada e a respetiva sequência em que ocorre.

Na Tabela 9, é apresentado um exemplo da primeira matriz. Como se pode observar, o número de colunas corresponde à soma de todas as operações dos trabalhos, enquanto o número de linhas é igual à soma de todas as máquinas capazes de executar essas operações. No exemplo,

consideram-se dois trabalhos, cada um com duas operações, resultando numa matriz com quatro colunas.

Tabela 9 – Exemplo da primeira matriz

Máquina	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$
0	200	0	200	0
1	0	100	0	150
2	0	100	0	50

A Tabela 10 mostra um exemplo da segunda matriz. O número de colunas desta matriz é calculado a partir do número de máquinas capazes de realizar cada operação. O total de colunas é a soma desses valores. No exemplo, apenas uma máquina pode realizar as operações $O_{1,1}$ e $O_{2,1}$, enquanto duas máquinas estão disponíveis para as operações $O_{1,2}$ e $O_{2,2}$. Portanto, o número de colunas da matriz é a 6.

Tabela 10 – Exemplo da segunda matriz

	$O_{2,1}$	$O_{2,2}$	$O_{1,1}$	$O_{2,2}$	$O_{1,2}$	$O_{1,2}$
Trabalho	2	2	1	2	1	1
Máquina	0	2	0	1	1	2

4.2.2 População Inicial

A população inicial é gerada conforme o pseudo-código apresentado no Algoritmo 1, que detalha todos os passos do algoritmo.

Algoritmo 1: Criação da população inicial

```
Definir o tamanho da população (pb ←  100)
Definir λ (λ ← 0,3)
Enquanto pb > 0
   pb ← pb – 1
   matriz1 ← criar matriz (número de máquinas (m), número total de operações de
   todos    os trabalhos (q))
   Para cada i em trabalhos
    Para cada j em operações do trabalho i
      TMSij ←  número de máquinas capazes de realizar a operação(j)
      lista_máquinas  ←  lista de máquinas capazes de realizar a operação(j)
      Ei  ←tamanho do lote do trabalho (i)
      Se TMSij == 1 então
        matriz1[máquina][i,j] ← Ei
      Senão se TMSij > 1 então
         número_aleatório ← gerar número aleatório ()
         Se número_aleatório < λ então
            máquina_selecionada ←  selecionar máquina aleatória(lista_máquinas)
            matriz1[máquina_selecionada][i,j] ← Ei
```

```
            Senão
               hi ← tamanho mínimo do lote do trabalho (i)
               máquina_selecionada ←  selecionar máquina aleatória(lista_máquinas)
               σijk ← gerar valor aleatório entre (hi, Ei - hi)
               matriz1[máquina_selecionada] [i,j] ← σijk
               TMSij ← TMSij – 1
               remover (lista_máquinas, máquina_selecionada)
               Enquanto TMSij - 2 > 0
                  μ ←  gerar valor aleatório entre (hi, Ei - hi - σijk)
                  máquina_selecionada ←  selecionar máquina
                  aleatória(lista_máquinas)
                  matriz1[máquina_selecionada] [i,j] ← μ
                  σijk ← σijk + μ
                  TMSij ← TMSij – 1
                  remover (lista_máquinas, máquina_selecionada)
               Fim Enquanto
               máquina_selecionada ←  selecionar máquina aleatória(lista_máquinas)
               matriz1[máquina_selecionada] [i,j] ← Ei – σijk
            Fim Se
         Fim Se
       Fim Para
     Fim Para
     matriz2  ←  criar matriz (2, soma das máquinas para as quais as operações
   podem ser processadas)
     #Para cada trabalho somar o número de máquinas capazes de realizar cada
   operação do trabalho
     Para cada i em trabalhos
       Para cada j em operações do trabalho i
         TMSi  ←  TMSi+ TMSij
       Fim para
     Fim para
     Para k de 1 até TMSi
       Adicionar i à lista_trabalhos
     Fim para
     lista trabalhos baralhada ← baralhar (lista_trabalhos)
     matriz2 [1] ← lista_trabalhos_baralhada
     lista_máquinas ← criar lista máquinas (trabalhos, operações)
     Para cada operação em lista_trabalhos_baralhada
       máquina_selecionada←selecionar máquina da lista (lista_máquinas)
       matriz2[2] [operação] ← máquina_selecionada
       remover (lista_máquinas, máquina_selecionada)
     Fim para
Fim Enquanto
```

4.2.3 Função Aptidão

O objetivo do problema em estudo é minimizar o *makespan* (C_{max}) que representa o tempo total necessário para concluir todos os trabalhos. O valor de C_{max} é obtido pelo conjunto de restrições definido pela equação (20).

$$C_{max} \geq C_{i,j,k} \quad \forall_{i,j,k,j \leq N_i, k \epsilon M_j} \tag{20}$$

O cálculo da função aptidão, que corresponde à função objetivo do problema, segue um procedimento específico. Primeiro, para cada coluna da segunda matriz, verifica-se em qual máquina a operação correspondente foi realizada. Em seguida, na primeira matriz, verifica-se

se o tamanho do lote dessa operação, para essa máquina, é maior que zero. Caso seja, o tempo de conclusão da última operação realizada na mesma máquina e o tempo de conclusão da operação precedente do trabalho em análise são considerados. O maior desses tempos é denominado "Tempo Inicial".

Com base nisso, o tempo de conclusão de uma operação é determinado pela fórmula apresentada na equação (21):

$$C_{i,j,k} = Tempo\ Inicial + \sigma_{i,j,k} * P_k + S_k \quad (21)$$

Nesta fórmula P_k representa o tempo de processamento da máquina k, S_k é o tempo de *setup* associado, e $\sigma_{i,j,k}$ indica o tamanho do sublote do trabalho i na operação j na máquina k.

4.2.4 Algoritmo de Pesquisa Local

Os passos detalhados do algoritmo de pesquisa local (APL) encontram-se descritos no Algoritmo 2. Este algoritmo é utilizado para aumentar a diversidade de soluções, considerando várias hipóteses para o tamanho dos sublotes nas diferentes máquinas.

Algoritmo 2: Algoritmo de Pesquisa Local

```
S1← selecionar cromossoma(população_atual)
Para cada operação Oij:
   Para cada máquina k em máquinas:
      Se σijk > 0 então
         Registar valor (σijk)
         máquina←k
         Parar loop
      Fim Se
   Fim Para
   hi ← tamanho mínimo do lote do trabalho (i)
   Ei ← tamanho do lote do trabalho (i)
   lista_máquinas ← máquinas capazes de realizar a operação (i,j)
   remover (lista_máquinas, máquina)
   Se σijk == Ei e TMSij>1 então
      matriz1[maquina][i,j] ← σijk-hi
      maquina_menor←selecionar máquina com menor tempo de processamento
      (lista_máquinas)
      matriz1[maquina_menor][i,j] ← hi
      S2 = nova_solução_obtida()
      FS2 = calcular valor de fitness(S2)
      Se FS2 <= FS1 então
         S1 ← S2
         FS1 ← FS2
      Fim Se
   Fim Se
   β ← quantidade a aumentar ou diminuir ao tamanho do sublote
   Se hi + β <= σ ijk <= Ei – hi então
     matriz1[maquina][i,j] ← σijk – β
     máquina_aleatória ← selecionar máquina aleatória(lista_máquinas)
     matriz1[máquina_aleatória][i,j] ← σijk + β
```

```
        S2 ← nova solução obtida ()
        FS2 ← calcular valor de fitness(S2)
        Se FS2 <= FS1:
            S1 ← S2
            FS1 ← FS2
        Fim Se
        Se FS2 < FS1:
            c ← 1
        Fim Se
        Enquanto FS2 <= FS1
            matriz1[maquina][i,j] ← σijk – β
            máquina_aleatória ← selecionar máquina aleatória(lista_máquinas)
            matriz1[máquina_aleatória][i,j] ← σijk + β
            S2 ← nova solução obtida ()
            FS2 ← calcular valor de fitness(S2)
            Se FS2 <= FS1 então
              S1 ← S2
              FS1 ← FS2
            Fim Se
            Se FS2 < FS1:
              c = 1
            Fim Se
            Se FS2 > FS1:
              Parar loop
            Fim Se
        Fim Enquanto
      Fim Se
   Se hi <= σijk <= Ei - hi – β então
      matriz1[maquina][i,j] ← σijk + β
      máquina_aleatória ← selecionar máquina aleatória(lista_máquinas)
      matriz1[máquina_aleatória][i,j] ← σijk – β
      S2 ←nova_solução_obtida()
      FS2 ← calcular valor de aptidão(S2)
      Se c == 1 e FS2 < FS1 então
         S1 ← S2
         FS1 ← FS2
         Enquanto FS2 < FS1
            matriz1[maquina][i,j] ← σijk + β
            máquina_aleatória ← selecionar máquina aleatória(lista_máquinas)
            matriz1[máquina_aleatória][i,j] ← σijk – β
            S2 ← nova_solução obtida ()
            FS2 ← calcular valor de aptidão(S2)
            Se FS2 >= FS1 então
               Parar loop
            Fim Se
          Fim Enquanto
      Senão se c ≠ 1 e FS2 <= FS1 então
            S1 ← S2
            FS1 ← FS2
            Enquanto FS2 <= FS1
               matriz1[maquina][i,j] ← σijk + β
               máquina_aleatória ← selecionar_máquina_aleatória(lista_máquinas)
               matriz1[máquina_aleatória][i,j] ← σijk – β
               S2 ← nova solução obtida()
               FS2 ← calcular valor de aptidão(S2)
               Se FS2 > FS1:
                 Parar loop
               Fim Se
            Fim Enquanto
       Fim Se
    Fim Se
Fim Se
```

```
Fim Para
Para cada máquina k em máquinas:
    matriz1[k][i,j] ← Ei
    Para cada máquina k1 em máquinas
      Se k1≠k então
         matriz1[k1][i,j] ← 0
      Fim Se
    Fim Para
    C_max ← calcular valor de fitness(solução_obtida)
    Se C_max < menor_Cmax então
      menor_Cmax←C_max
      menor_S2← S2
    Fim Se
    S2←menor_S2
    FS2 ← menor_Cmax
    Se FS2 <= FS1 então
      S1 ← S2
      FS1 ← FS2
    Fim Se
Fim Para
```

4.2.5 Seleção, Cruzamento e Mutação

Os algoritmos genéticos são constituídos pelos seguintes componentes: o mecanismo de seleção e o mecanismo de exploração do espaço de procura, geralmente através dos operadores genéticos.

A seleção visa manter os indivíduos com melhor desempenho na população e eliminar os de pior desempenho. No AGH proposto são aplicados dois métodos de seleção: elitismo e seleção por torneio binário.

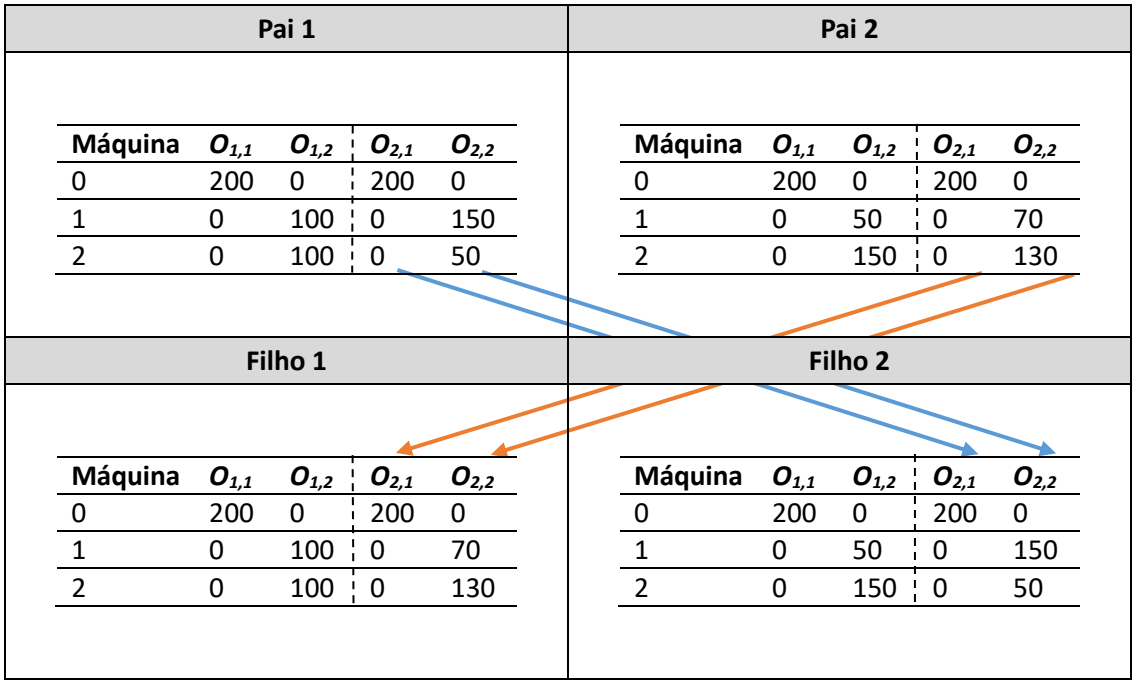
1. Elitismo: Nesta técnica, uma percentagem predeterminada da população, constituída pelos melhores indivíduos encontrados até ao momento, forma a elite. Deste modo, garante-se que os melhores indivíduos encontrados estão presentes na última geração, evitando que soluções de qualidade sejam perdidas ao longo das iterações (Da Costa, 2003).

2. Seleção por torneio binário: Este método seleciona dois indivíduos aleatoriamente da população para competir. O indivíduo com melhor aptidão é escolhido para passar para a próxima geração, permitindo que apenas os mais aptos sejam submetidos aos operadores genéticos.

Os operadores genéticos, correspondentes ao cruzamento e à mutação, são importantes na exploração do espaço de soluções, melhorando a diversidade da população essencial para uma pesquisa eficiente de soluções.

O cruzamento combina a informação genética de dois progenitores para gerar novos descendentes. No AGH, utilizam-se dois tipos de cruzamento: o cruzamento de um ponto e o cruzamento JBX**.**

O cruzamento de um ponto é aplicado à primeira matriz. Um ponto de corte é selecionado aleatoriamente, e as informações genéticas dos pais são trocadas a partir desse ponto. Na Figura 5, apresenta-se um exemplo em que o ponto de corte é a coluna 2.



Máquina	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$
0	200	0	200	0
1	0	100	0	150
2	0	100	0	50

Máquina	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$
0	200	0	200	0
1	0	50	0	70
2	0	150	0	130

Máquina	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$
0	200	0	200	0
1	0	100	0	70
2	0	100	0	130

Máquina	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$
0	200	0	200	0
1	0	50	0	150
2	0	150	0	50

Figura 5 – Exemplo do método de cruzamento de um ponto

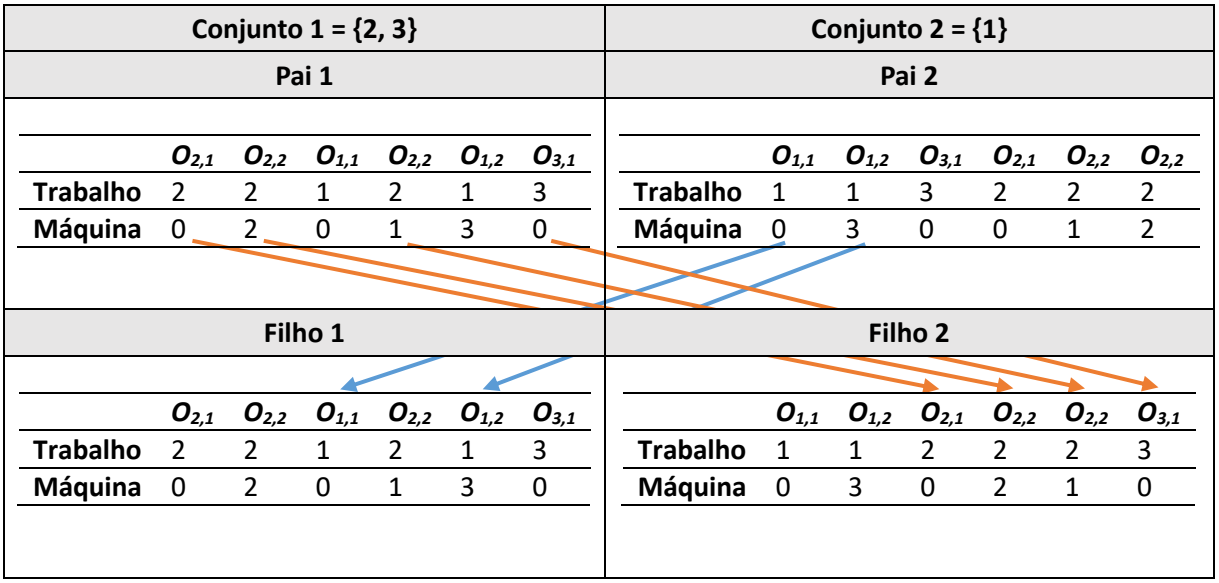
O método JBX é aplicado à segunda matriz e segue o seguinte procedimento (Li e Gao, 2016):

1. Divide-se o conjunto de trabalhos em dois grupos: Conjunto 1 e Conjunto 2, de forma aleatória.

2. Os elementos do Pai 1 pertencentes ao Conjunto 1 são mantidos na mesma posição no Filho 1, e os elementos do Pai 2 pertencentes ao Conjunto 2 são mantidos no Filho 2. Os restantes elementos são excluídos dos pais.

3. Os elementos restantes do Pai 2 são atribuídos às posições vazias do Filho 1, e os restantes do Pai 1 vão para as posições vazias do Filho 2.

Na Figura 6, é apresentado um exemplo da aplicação do método JBX.



	$O_{2,1}$	$O_{2,2}$	$O_{1,1}$	$O_{2,2}$	$O_{1,2}$	$O_{3,1}$
Trabalho	2	2	1	2	1	3
Máquina	0	2	0	1	3	0

	$O_{1,1}$	$O_{1,2}$	$O_{3,1}$	$O_{2,1}$	$O_{2,2}$	$O_{2,2}$
Trabalho	1	1	3	2	2	2
Máquina	0	3	0	0	1	2

	$O_{2,1}$	$O_{2,2}$	$O_{1,1}$	$O_{2,2}$	$O_{1,2}$	$O_{3,1}$
Trabalho	2	2	1	2	1	3
Máquina	0	2	0	1	3	0

	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,2}$	$O_{3,1}$
Trabalho	1	1	2	2	2	3
Máquina	0	3	0	2	1	0

Figura 6 – Exemplo do método de cruzamento JBX

A mutação introduz variações aleatórias nas soluções, permitindo a exploração de novas direções no espaço de procura. Neste AGH, utiliza-se um mecanismo baseado em dois processos: inversão e SWAP, controlados por um número aleatório r. Se $r > a$, aplica-se a inversão; caso contrário, utiliza-se o SWAP.

No processo de inversão, dois pontos são selecionados aleatoriamente, e a ordem dos elementos entre eles é invertida. Um exemplo deste processo é ilustrado na Figura 7.

	$O_{2,1}$	$O_{2,2}$	$O_{1,1}$	$O_{2,2}$	$O_{1,2}$	$O_{3,1}$
Trabalho	2	2	1	2	1	3
Máquina	0	2	0	1	3	0

	$O_{2,1}$	$O_{1,2}$	$O_{2,2}$	$O_{1,1}$	$O_{2,2}$	$O_{3,1}$
Trabalho	2	1	2	1	2	3
Máquina	0	3	1	0	2	0

Figura 7 – Exemplo de aplicação do método de inversão

No método SWAP, dois pontos são selecionados aleatoriamente e os seus valores são trocados. Um exemplo da sua aplicação, é apresentado na Figura 8.

	$O_{2,1}$	$O_{2,2}$	$O_{1,1}$	$O_{2,2}$	$O_{1,2}$	$O_{3,1}$
Trabalho	2	2	1	2	1	3
Máquina	0	2	0	1	3	0

	$O_{2,1}$	$O_{1,2}$	$O_{1,1}$	$O_{2,2}$	$O_{2,2}$	$O_{3,1}$
Trabalho	2	1	1	2	2	3
Máquina	0	3	0	1	2	0

Figura 8 – Exemplo de aplicação do método SWAP

Após a aplicação da mutação, é necessária a correção da segunda matriz, visto que a ordem das operações pode ter sido alterada de forma inadequada. Por exemplo, na Figura 8, a operação $O_{1,2}$ aparece antes de $O_{1,1}$, o que não é possível. A correção é realizada conforme ilustrado na Figura 9.

	$O_{2,1}$	$O_{1,2}$	$O_{1,1}$	$O_{2,2}$	$O_{2,2}$	$O_{3,1}$
Trabalho	2	1	1	2	2	3
Máquina	0	3	0	1	2	0

	$O_{2,1}$	$O_{1,1}$	$O_{1,2}$	$O_{2,2}$	$O_{2,2}$	$O_{3,1}$
Trabalho	2	1	1	2	2	3
Máquina	0	0	3	1	2	0

Figura 9 – Correção da segunda matriz

4.2.6 Critério de Paragem

O critério de paragem do algoritmo foi definido com base nas seguintes condições:

- **Número máximo de gerações**: A execução do algoritmo é interrompida quando o número máximo de gerações atinge 200.

- **Convergência da solução**: O algoritmo também pode parar se o erro entre a solução atual e a solução anterior for inferior a 5% e essa condição for atendida pelo menos 10 vezes consecutivas.

4.2.7 Desenvolvimento de um Sistema de Apoio à Decisão

Para o desenvolvimento do programa que implementa o AGH, foi utilizada a linguagem Python. O sistema foi projetado para processar os dados de entrada (input) relacionados aos trabalhos a serem escalonados e os parâmetros específicos do AGH. O principal resultado gerado é o escalonamento otimizado dos trabalhos. Além disso, o programa produz um Gráfico de *Gantt* que visualiza o cronograma dos trabalhos, levando em consideração a disponibilidade diária de cada máquina.

A Figura 10 ilustra o fluxograma do procedimento seguido pelo sistema, detalhando as etapas desde a leitura dos dados até a geração do Gráfico de *Gantt*.

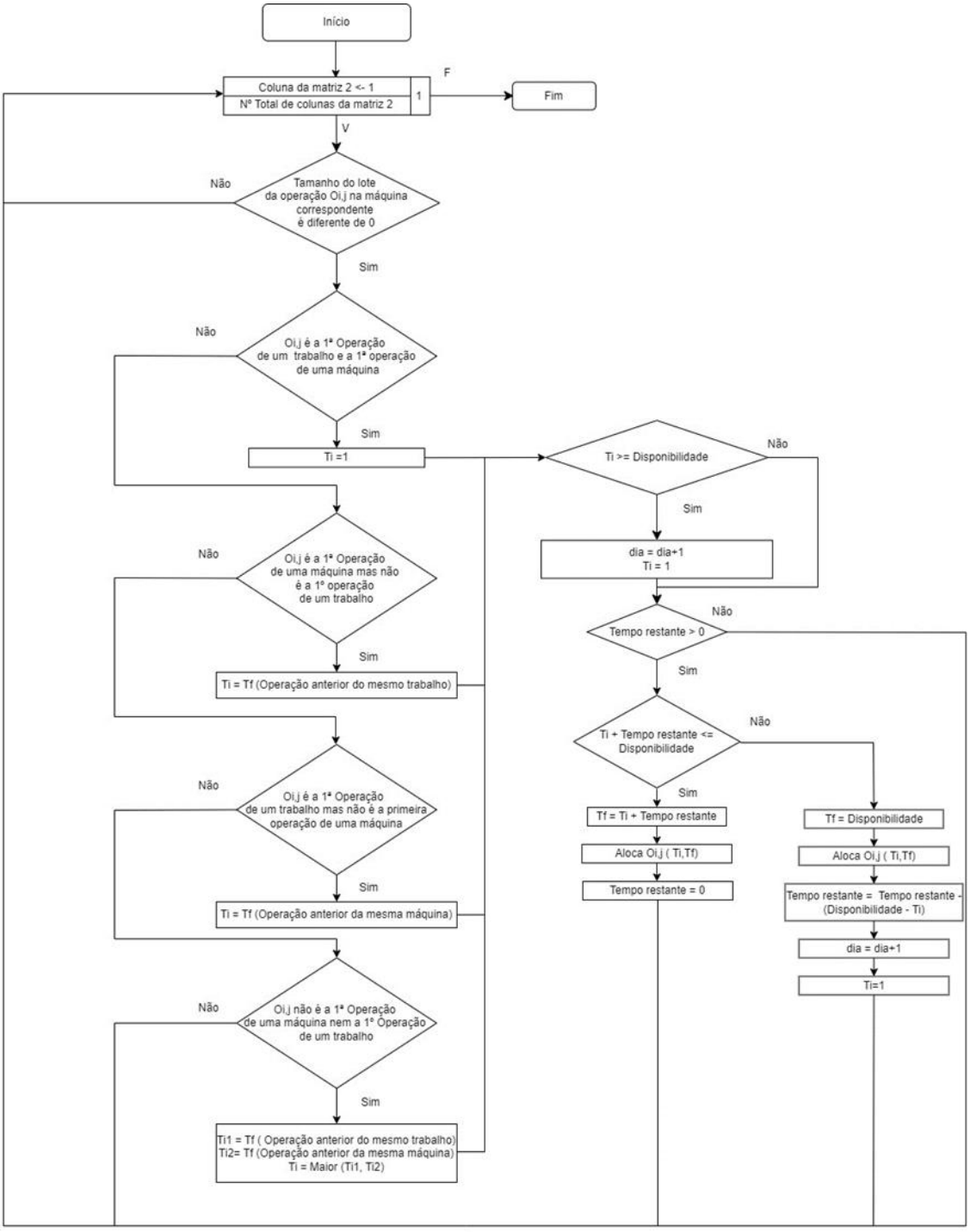


Figura 10 – Fluxograma do sistema de apoio à decisão com abordagem *job-splitting*

4.3 Experiências Computacionais

O AGH e o sistema de apoio à decisão foram implementados Python 3.11.4. Os testes foram realizados num computador com Intel Core i5 de 2,40 GHz e 8 GB de RAM.

4.3.1 Determinação dos Parâmetros do AGH

4.3.1.1 Tamanho da população

Com o objetivo de escolher o valor ideal para o tamanho da população, foram conduzidos vários testes. Nesses testes, analisou-se a variação do tempo *Central processing unit* (CPU) (Figura 11) e do *makespan* (Figura 12) em função do aumento do tamanho da população, utilizando duas instâncias diferentes. Ambas as instâncias são constituídas por três trabalhos, a primeira com três operações cada um e a segunda com cinco operações. Os valores apresentados no gráfico correspondem à média de dez testes realizados para cada valor do tamanho da população. Todos os resultados estão disponíveis no Apêndice A.

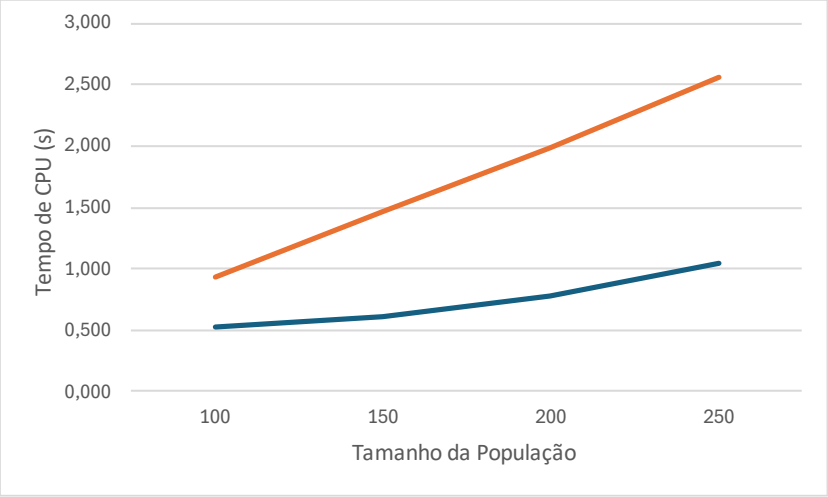


Figura 11 – Variação do tempo CPU (s) em função do tamanho da população

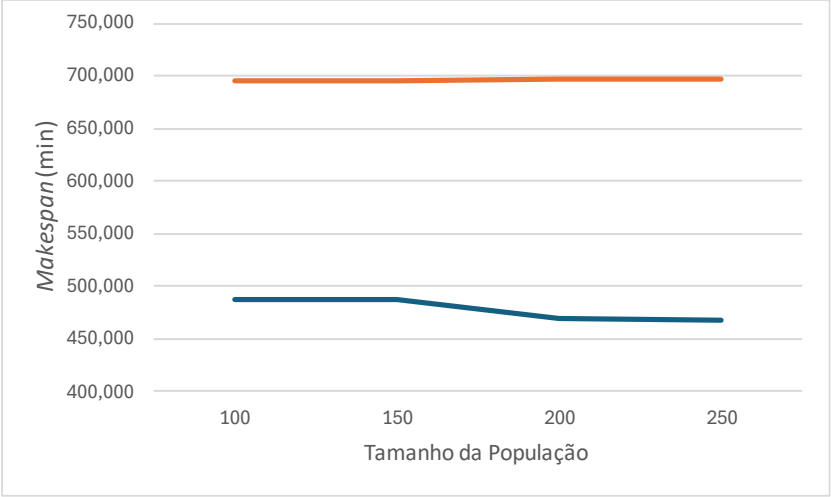


Figura 12 – Variação do *makespan* (min) em função do tamanho da população

A análise da Figura 11 revela uma relação direta entre o tamanho da população e o tempo de CPU: à medida que o tamanho da população aumenta, o tempo computacional e as exigências de memória também crescem. Já na Figura 12, verifica-se que não há uma correlação clara entre o tamanho da população e o valor do *makespan*. O *makespan* não diminui com o aumento da população, apresentando pouca variação e permanecendo relativamente estável para os diferentes tamanhos testados.

Com base nestes resultados, optou-se por definir o tamanho da população em 100, equilibrando a eficiência computacional com a qualidade dos resultados obtidos. Esta escolha mostrou-se proeficiente para a resolução de instâncias de grandes dimensões.

4.3.1.2 Taxa de cruzamento

A taxa de cruzamento é um parâmetro que gere a probabilidade de um cruzamento ocorrer entre duas soluções. Quanto maior o seu valor, maior é a probabilidade de ocorrer o cruzamento, gerando novas soluções (descendentes) que combinam as melhores características dos pais. Para determinar o valor deste parâmetro, analisou-se o comportamento do tempo CPU (Figura 13) e do *makespan* (Figura 14) em função da variação da taxa de cruzamento, utilizando duas instâncias diferentes. Os valores nos gráficos correspondem à média de dez testes realizados para cada valor da taxa de cruzamento. Os resultados detalhados encontram-se no Apêndice B.

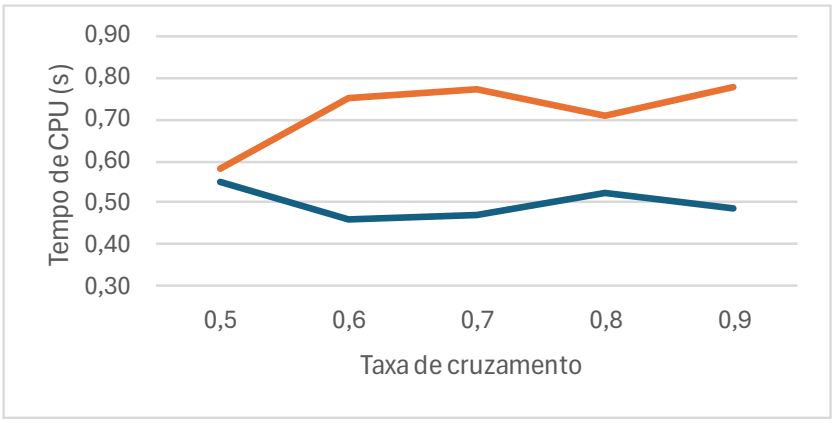


Figura 13 – Variação do tempo CPU (s) em função do valor da taxa de cruzamento

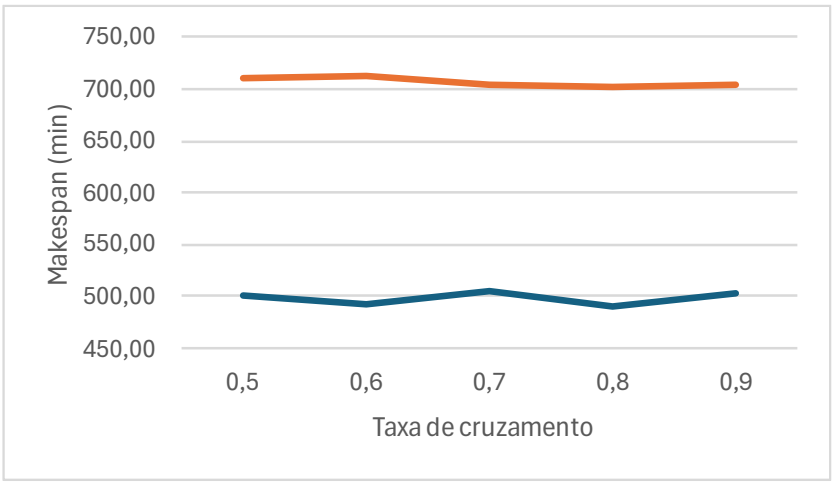


Figura 14 – Variação do *makespan* (min) em função do valor da taxa de cruzamento

A análise da Figura 13 mostra que o tempo de CPU não tem um comportamento uniforme entre as instâncias. Na instância 1 (representada a azul), a taxa de cruzamento que resulta no menor tempo de CPU é 0,6, enquanto na instância 2 (representada a laranja), esse valor é 0,5. Em relação ao *makespan* (Figura 14), observa-se que a taxa de cruzamento com os melhores resultados é 0,8. Perante estes resultados, optou-se por definir a taxa de cruzamento em 0,8. Além disso, o estudo de Liangxiao e Zhongjun (2015) também identifica 0,8 como a taxa mais eficaz nos testes que efetuaram, reforçando esta escolha.

4.3.1.3 Parâmetro β

Para definir o valor ideal do parâmetro β, foram realizados vários testes, nos quais se analisou o comportamento do tempo de CPU (Figura 15) e do *makespan* (Figura 16) em função da variação desse parâmetro. Foram conduzidos 10 testes computacionais em duas instâncias

distintas, para cada valor de β. Os gráficos apresentam a média dos resultados obtidos, conforme detalhados no Apêndice C.

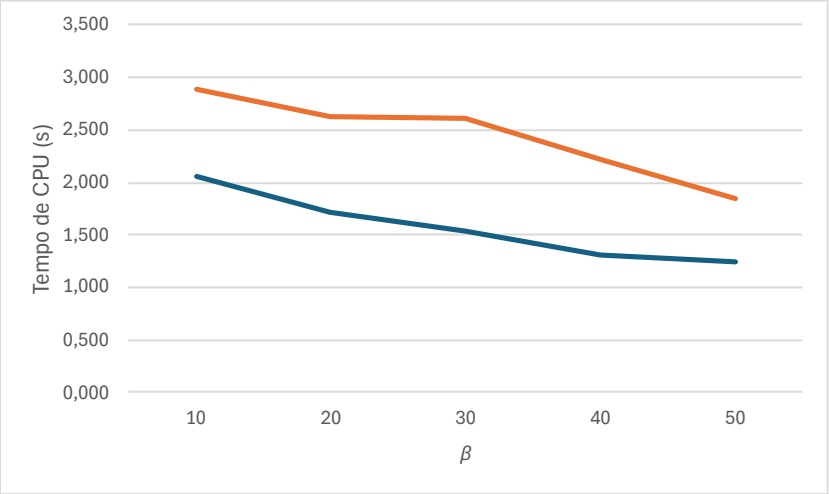


Figura 15 - Variação do tempo CPU (s) em função do valor do parâmetro β

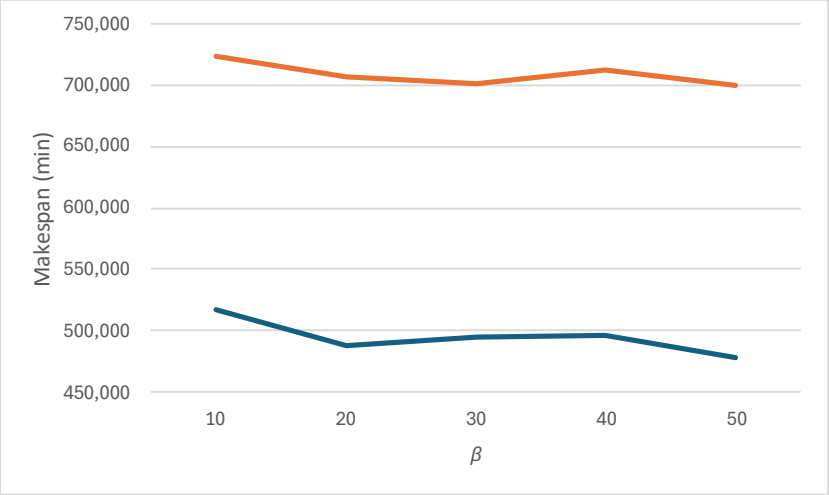


Figura 16 - Variação do *makespan* (min) em função do valor do parâmetro β

A análise da Figura 15 revela uma diminuição contínua do tempo de CPU à medida que o valor de β aumenta, o que pode ser explicado pela redução no número de testes necessários ao utilizar o APL. Já na Figura 16, observa-se que o makespan se mantém relativamente estável com a variação de β, sendo que o valor de β que apresenta o menor *makespan* para ambas as instâncias é 50. Assim, o valor de β escolhido foi 50, por apresentar o menor tempo de CPU e o menor *makespan* nos testes realizados.

4.3.1.4 Restantes parâmetros

O tamanho da população, a taxa de cruzamento e o valor de β foram escolhidos através da análise de vários testes, apresentados nas subseções anteriores. Os parâmetros p, taxa de mutação e a foram retirados dos estudos de Tutumlu e Saraç (2023).

Os valores dos parâmetros do AGH encontram-se na Tabela 11.

Tabela 11 – Parâmetros do AGH

β	Tamanho da população	p	Taxa Cruzamento	Taxa Mutação	a
50	100	25	80%	5%	0,5

4.3.2 Descrição de uma Instância de Teste e Resultados Obtidos pelo AGH

Nesta seção apresenta-se uma instância de teste e os resultados obtidos pelo AGH. A instância de teste envolve um conjunto de trabalhos, cada um composto por uma ou mais operações, e os respetivos postos de trabalho para executar essas operações.

A Tabela 12 apresenta os 2 trabalhos que compõem a instância de teste e as respetivas operações. Cada trabalho i é composto por um conjunto de operações, $O_{i,j}$, onde o j representa a ordem da operação. Para cada operação, é especificada a quantidade a ser produzida e o posto de trabalho responsável pela produção.

Tabela 12 – Exemplo de uma instância de teste

Trabalho	Operação	Quantidade	Posto de Trabalho
1	$O_{1,1}$	200	PC
1	$O_{1,2}$	200	EM
1	$O_{1,3}$	200	SL
2	$O_{2,1}$	200	PC
2	$O_{2,2}$	200	KG7

A Tabela 13 apresenta os dados dos postos de trabalho, incluindo a disponibilidade das máquinas, a identificação das máquinas (id), o tempo de *setup* e o tempo de processamento por unidade.

Tabela 13 – Dados dos postos de trabalho

Posto de trabalho	Disponibilidade (min)	Máquina (id)	*Setup* (min)	Tempo (min)
PC	450	26	5	0,24
EM	1350	20	10	1
SL	450	18	2	0,19
KG7	1440	6	10	4,33
		7	10	5
		8	10	5

Após a aplicação do AGH, os resultados incluem a atribuição dos trabalhos às máquinas e a quantidade produzida por cada máquina. A Tabela 14 resume os resultados obtidos, onde se pode visualizar o tamanho de sublote de cada operação em cada máquina.

Tabela 14 – Atribuição dos trabalhos às máquinas e a quantidade produzida por máquina

Máquina	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$
26	200	0	0	200	0
20	0	200	0	0	0
18	0	0	200	0	0
6	0	0	0	0	80
7	0	0	0	0	70
8	0	0	0	0	50

Na Figura 17, encontra-se representada de forma esquemática para uma melhor compreensão a solução encontrada pelo AGH.

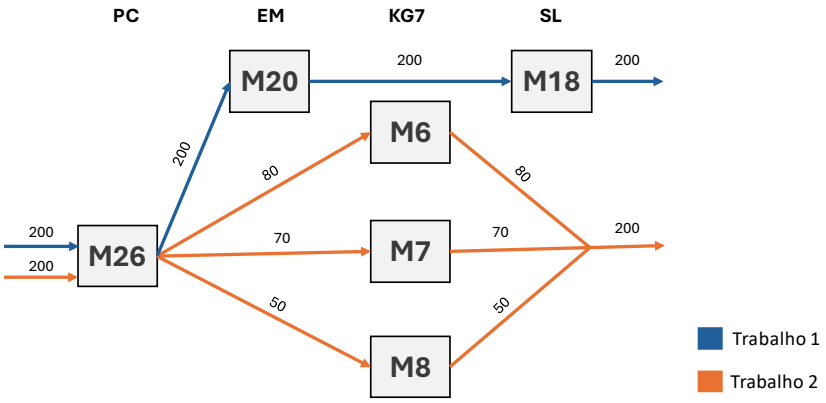


Figura 17 – Representação gráfica da solução encontrada pelo AGH

A Tabela 15, mostra a segunda matriz da solução encontrada, que indica a sequência das operações.

Tabela 15 – Segunda matriz da solução da instância de teste

Operação	$O_{2,1}$	$O_{1,1}$	$O_{2,2}$	$O_{2,2}$	$O_{1,2}$	$O_{1,3}$	$O_{2,2}$
Trabalho	2	1	2	2	1	1	2
Máquina	26	26	7	6	20	18	8

Com base nesta matriz é possível apresentar a sequência de operações para cada máquina, tal como se apresenta na Tabela 16.

Tabela 16 – Sequência de operações para cada máquina

Máquina	**Sequência de Operações**
26	$O_{2,1} \rightarrow O_{1,1}$
20	$O_{1,2}$
18	$O_{1,3}$
6	$O_{2,2}$
7	$O_{2,2}$
8	$O_{2,2}$

Quando há divisão de trabalhos, como é o caso da operação $O_{2,2}$, o tempo de conclusão é determinado pelo máximo dos tempos de conclusão das máquinas que executam essa operação, garantindo que a operação só é considerada concluída quando a última máquina terminar o processamento. De acordo com a Tabela 17, sabendo que o tempo de conclusão da da operação $O_{2,1}$ é igual a 53 minutos, o tempo de conclusão da operação $O_{2,2}$ é de 414 minutos.

Tabela 17 - Cálculo do tempo de conclusão do sublote da operação $O_{2,2}$ (min)

Máquina	**Tempo de conclusão do sublote da operação $O_{2,2}$ (min)**
6	53+1+80×4,33+10 = 410,4
7	53+1+70×5+10 = 414
8	53+1+50×5+10 = 314

Na Figura 18, é apresentado o gráfico de *Gantt* com o escalonamento dos dois trabalhos nas máquinas, incluindo a divisão do trabalho 2 em três sublotes na segunda operação.

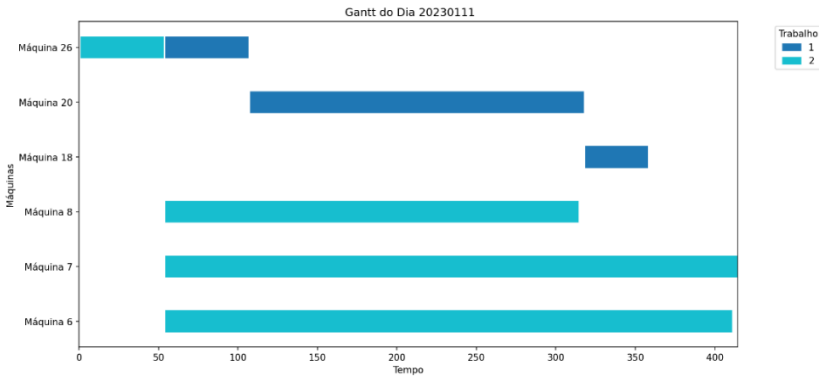


Figura 18 – Gráfico de *Gantt* com o escalonamento dos trabalhos

4.3.3 Análise dos Tempos Computacionais em Função da Variação do Número de Operações e Trabalhos nas Instâncias de Teste

Na presente secção, analisam-se os tempos computacionais para diferentes instâncias de teste. O objetivo foi avaliar o desempenho do AGH em relação ao aumento da complexidade do problema. Para cada instância de teste, variou-se o número de trabalhos e de operações por trabalho, mantendo constantes outros parâmetros do problema, como a quantidade a produzir. Os tempos computacionais são registados para cada configuração, com o intuito de analisar como o aumento do número de operações e trabalhos afeta o tempo de execução do AGH. Na Tabela 18, são apresentadas algumas características de cada instância testada e os respetivos valores do tempo de CPU obtidos. Os valores apresentados resultam da escolha do tempo de CPU correspondente ao teste com menor *makespan*, de entre um conjunto de 10 testes realizados (Apêndice D).

Tabela 18 - Instâncias teste e respetivo tempo de CPU (s)

Problema	**Nº Trabalhos**	**Nº Operações por trabalho**	**Quantidade por trabalho**	**Nº máquinas no total**	**Nº máximo de máquinas por posto**	**Tempo de CPU (s)**
Instância 1	2	2	200	7	3	0,58
Instância 2	2	3	200	9	3	1,09
Instância 3	2	4	200	9	3	0,94
Instância 4	2	5	200	10	3	1,69
Instância 5	2	6	200	14	3	2,67
Instância 6	2	7	200	15	3	3,19
Instância 7	2	8	200	16	3	2,30
Instância 8	2	9	200	19	3	3,55
Instância 9	3	1	200	7	3	0,33

Instância 10	3	2	200	10	3	0,58
Instância 11	3	3	200	14	3	1,23
Instância 12	3	4	200	15	3	2,13
Instância 13	3	5	200	16	3	3,34
Instância 14	3	6	200	19	3	3,13

Os testes revelam que o aumento da complexidade do problema, ou seja, o número de trabalhos e operações, têm impacto no tempo de CPU.

4.3.4 Desempenho do AGH em Instâncias do Problema Real: Análise dos Tempos de CPU para Diferentes Dimensões

Com os dados da indústria têxtil que lançou o desafio, foram criadas instâncias de teste com três dimensões distintas: pequena, média e grande. O tempo de CPU para cada uma das instâncias está detalhado na Tabela 19. Observando a tabela, verifica-se que para a instância grande com 67 trabalhos, o tempo de CPU registado é de aproximadamente 35 minutos. Os resultados sugerem que a meta-heurística pode ser efetivamente utilizada em cenários reais, fornecendo soluções em tempo útil e admissíveis para problemas complexos como o escalonamento na fabricação de artigos desportivos.

Tabela 19 - Instâncias reais da indústria e respetivo tempo de CPU (s)

Problema	**Número Trabalhos**	**Nº Máximo de Operações por Trabalho**	**Quantidade máxima por trabalho**	**Nº total de máquinas**	**Nº máximo de máquinas por posto**	**Tempo de CPU (s)**
Instância Pequena	7	8	800	25	3	11,41
Instância Média	57	8	1500	32	3	1147,67
Instância Grande	67	8	3500	32	3	2145,50

Além destes testes, aplicou-se o algoritmo a uma instância com 200 trabalhos e conseguiu-se obter uma solução num tempo computacional de aproximadamente 16 horas.

Nas Figuras 19, 20 e 21 são apresentados os gráficos de *Gantt* gerados pelo sistema de apoio à decisão, que mostram o escalonamento de 7 trabalhos da "Instância Pequena". Como mencionado anteriormente, o sistema considera a disponibilidade diária das máquinas, estando indexado ao tempo. A análise dos diagramas revela que são necessários 3 dias para atender todas as ordens produção.

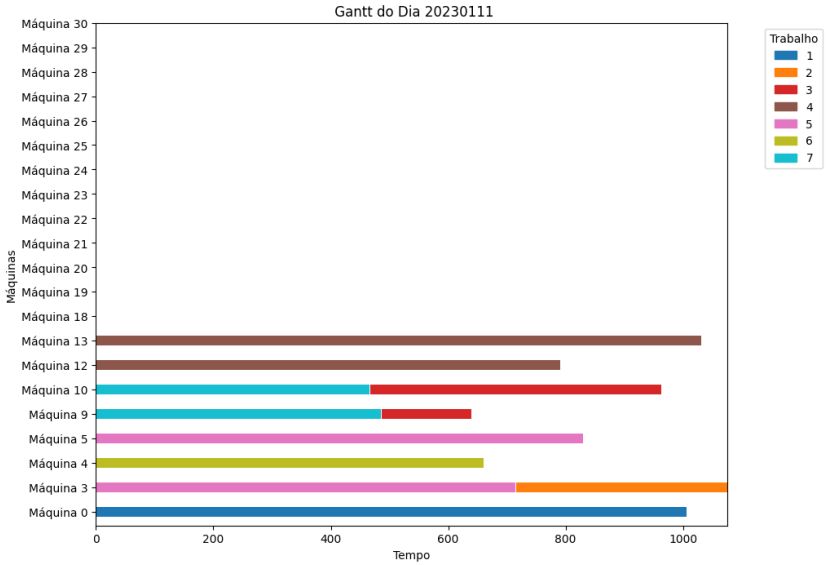


Figura 19 – Gráfico de *Gantt* para a Instância Pequena (Dia 1)

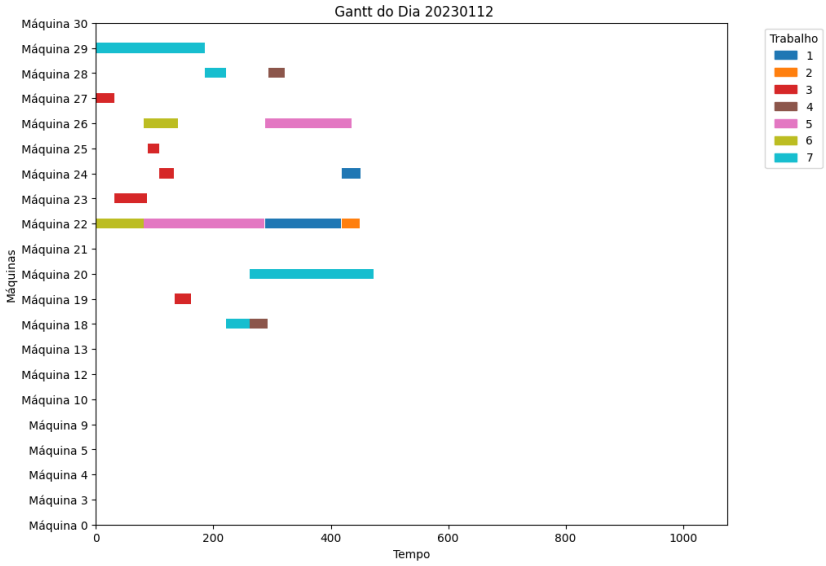


Figura 20 - Gráfico de *Gantt* para a Instância Pequena (Dia 2)

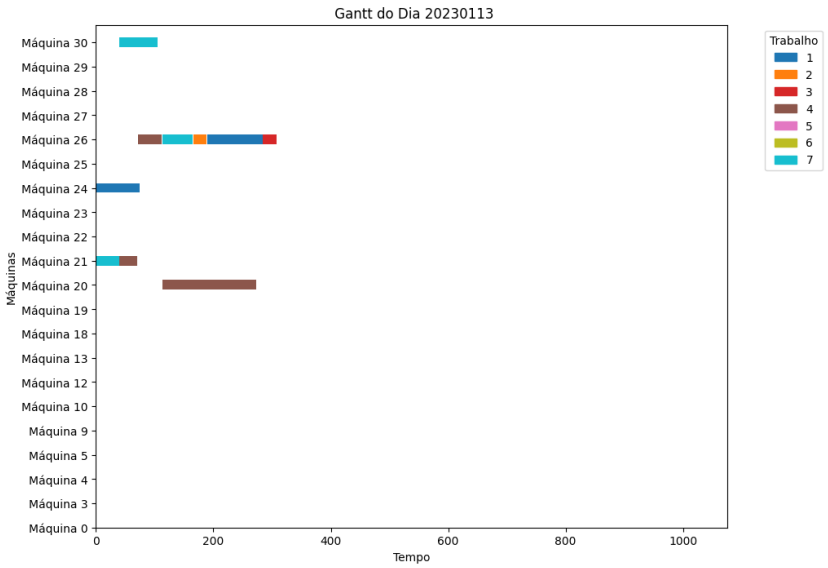


Figura 21 - Gráfico de *Gantt* para a Instância Pequena (Dia 3)

4.4 Desempenho do AGH *versus* o Modelo PLIM1

Nesta secção, são apresentados os valores do *makespan* obtidos pelo modelo PLIM1 e pela meta-heurística para as mesmas instâncias, bem como o erro calculado através da equação (22), representados na Tabela 20.

$$Erro = \frac{Makespan\ AGH - Makespan\ PLIM1}{Makespan\ PLIM1} \times 100 \qquad (22)$$

Tabela 20 – Comparação dos valores dos *makespan* obtidos pelo modelo PLIM1 e pelo AGH

Problema	**NºTrabalhos**	**NºOperações**	***Makespan*(min) PLIM1**	***Makespan*(min) AGH**	**Erro (%)**
Instância 1	2	2	361	373,50	3,46
Instância 2	2	3	402	415	3,23
Instância 3	2	4	446,50	461	3,25
Instância 4	2	5	499	514	3,01
Instância 5	2	6	2897,50	2982,50	2,93
Instância 6	2	7	2962,50	3047,50	2,87
Instância 7	2	8	3143,23	3193,90	1,61
Instância 8	2	9	3471	3592,40	3,50
Instância 9	3	1	475	513,67	8,14
Instância 10	3	2	382,33	414,33	8,37
Instância 11	3	3	402	427	6,22
Instância 12	3	4	610	644	5,57
Instância 13	3	5	648	683,33	5,45
Instância 14	3	6	713,33	749,33	5,05

Os valores apresentados na Tabela 20, resultaram da escolha do melhor resultado de entre 10 soluções testadas, que estão apresentadas no Apêndice D. A média do valor do erro corresponde a 4,48 %, ou seja, pode-se considerar que a meta-heurística encontra soluções próximas da solução ótima.

5 Modelo de Programação Linear Inteira Mista com *Lot Streaming*

Neste capítulo, é abordado o escalonamento do JSF com a técnica *lot streaming*. Primeiramente, é apresentada a formulação matemática de um modelo de Programação Linear Inteira Mista (PLIM2). Em seguida, é descrito o sistema de apoio à decisão. Por fim, são apresentadas as experiências computacionais realizadas.

5.1 Formulação Matemática

O problema em estudo pode ser descrito da seguinte forma. Considera-se um plano de produção composto por n trabalhos $I = \{I_1, I_2, \dots, I_n\}$, cada um com uma quantidade (ou ordem de produção) a produzir correspondente a $E = \{E_1, E_2, \dots, E_n\}$. Sabe-se que cada trabalho I_i possui N_i operações, que devem ser realizadas numa sequência previamente definida. Além disso, a quantidade a produzir E_i de I_i pode ser dividida em vários sublotes, com um limite máximo de B_i sublotes.

O modelo baseia-se nos seguintes pressupostos:

- As rotas dos trabalhos são fornecidas previamente;

- Cada máquina realiza exclusivamente uma operação;

- Todos os tempos de processamento e de preparação (*setup*) são determinísticos;

- Cada máquina pode processar apenas um sublote de cada vez;

- Cada sublote pode ter apenas uma operação em processamento de cada vez;

- A operação de um sublote pode ser interrompida se a disponibilidade da máquina não permitir a sua conclusão no dia em questão. Nestes casos, a operação será retomada no dia seguinte, e assim sucessivamente, até que a quantidade total a produzir esteja completa.

As Tabelas 21, 22, 23 e 24 apresentam, respetivamente, os conjuntos, os índices, os parâmetros e as variáveis de decisão do modelo.

De seguida, apresenta-se o modelo de Programação Linear Inteira Mista (PLIM) para o problema em questão. O modelo proposto foi adaptado do trabalho de Fan et al. (2023) para satisfazer os objetivos do presente estudo.

Tabela 21 – Conjuntos do PLIM2

Conjuntos	**Descrição**
$\boldsymbol{I = \{1, \dots, n\}}$	Conjunto de trabalhos
$\boldsymbol{W = \{1, \dots, c\}}$	Conjunto de postos de trabalho
$\boldsymbol{E = \{E_1, \dots, E_n\}}$	Conjunto com as quantidades de produção, onde E_i indica o volume da encomenda do trabalho i, com $i = 1, \dots, n$
$\boldsymbol{N = \{N_1, \dots, N_n\}}$	Conjunto com o número de operações por trabalho, onde N_i representa o número total de operações do trabalho i, com $i = 1, \dots, n$
$\boldsymbol{B = \{B_1, \dots, B_n\}}$	Conjunto com o número máximo de sublotes, onde B_i indica o número máximo de sublotes do trabalho i, com $i = 1, \dots, n$
$\boldsymbol{M_{k_s} = \{m_{s,1}, m_{s,2}, \dots, m_{s,m_s}\}}$	Conjunto de máquinas que pertencem ao posto de trabalho

Tabela 22 – Índices do PLIM2

Índices	**Descrição**
$\boldsymbol{i, i' \in I}$	Indicam um trabalho
$\boldsymbol{j, j' \in N_i}$	Indicam uma operação do trabalho i
$\boldsymbol{b, b' \in B_i}$	Indicam os sublotes do trabalho i
$\boldsymbol{s, s' \in W}$	Indicam os postos de trabalho
$\boldsymbol{k_s, k'_s \in M_{k_s}}$	Indicam as máquinas de cada posto de trabalho s

Tabela 23 – Parâmetros do PLIM2

Parâmetros	Descrição
$\boldsymbol{a_i}$	Vetor com os valores do número máximo de peças por repetição entre todas as máquinas do posto de trabalho onde é realizada cada operação j do trabalho i
$\boldsymbol{B_i}$	Número máximo de sublotes de um trabalho I_i $B_i = \frac{E_i}{Máximo\ (a_i)}$
$\boldsymbol{\phi_i}$	Número mínimo a produzir por sublote de I_i $\phi_i = int\left(\frac{E_i}{B_i}\right)$
$\boldsymbol{m_s}$	Número total de máquinas para o posto de trabalho W_s
$\boldsymbol{P_{s,k_s}}$	Tempo de processamento da máquina M_{k_s} do posto de trabalho W_s (min)
$\boldsymbol{S_{s,k_s}}$	Tempo de *setup* da máquina M_{k_s} do posto de trabalho W_s (min)
Ω	Um número positivo muito grande

Tabela 24 – Variáveis de Decisão do PLIM2

Variáveis de Decisão	Descrição
$\boldsymbol{C_{max}}$	*Makespan*
$\boldsymbol{\sigma_{i,b}}$	Tamanho do $b^{ésimo}$ sublote de I_i
$\boldsymbol{\delta_{i,b}}$	1, se o tamanho do $b^{ésimo}$ sublote de I_i é diferente de zero 0, caso contrário
$\boldsymbol{C_{i,j,b}}$	Tempo de conclusão de $\tau_{i,j,b}$
$\boldsymbol{X_{i,j,b,s,k_s}}$	1, se $\tau_{i,j,b}$ é processado na máquina M_{k_s} do posto de trabalho W_s 0, caso contrário
$\boldsymbol{\eta_{i,j,b,i',j',b'}}$	1, se $\tau_{i,j,b}$ é processado depois de $\tau_{i',j',b'}$ em uma máquina 0, caso contrário

A formulação matemática do problema é descrita a seguir.

$$\boldsymbol{minimizar\ f = C_{max}} \tag{23}$$

Sujeito a:

$$\sum_{b=1}^{B_i} \sigma_{i,b} \geq E_i \quad \forall_i \tag{24}$$

$$\delta_{i,b} * \Omega \geq \sigma_{i,b} \quad \forall_{i,b} \tag{25}$$

$$\delta_{i,b} \leq \sigma_{i,b} \quad \forall_{i,b} \tag{26}$$

$$\sigma_{i,b} \geq \phi_i \delta_{i,b} \quad \forall_{i,b} \tag{27}$$

$$\sum_{k_s=1}^{m_s} x_{i,j,b,s,k_s} = \delta_{i,b} \quad \forall_{i,j,b,s} \tag{28}$$

$$C_{i,j,b} \geq \sigma_{i,b} * P_{s,k_s} + S_{s,k_s} - \left(1 - x_{i,j,b,s,k_s}\right) * \Omega \quad \forall_{i,j=1,b,s,k_s} \tag{29}$$

$$C_{i,j,b} \geq C_{i,j-1,b} + \sigma_{i,b} * P_{s,k_s} + S_{s,k_s} - \left(1 - x_{i,j,b,s,k_s}\right) * \Omega \quad \forall_{i,j>1,b,s,k_s} \tag{30}$$

$$C_{i,j,b} \geq C_{i',j',b'} + \sigma_{i,b} * P_{s,k_s} + S_{s,k_s} - (3 - X_{i,j,b,s,k_s} - X_{i',j',b',s,k_s} - \eta_{i,j,b,i',j',b'}) * \Omega \quad \forall_{i,i',j,j',b,b',s,k_s, \tau_{i,j,b} \neq \tau_{i',j',b'}} \tag{31}$$

$$C_{i',j',b'} \geq C_{i,j,b} + \sigma_{i',b'} * P_{s,k_s} + S_{s,k_s} - (2 - X_{i,j,b,s,k_s} - X_{i',j',b',s,k_s} + \eta_{i,j,b,i',j',b'}) * \Omega \quad \forall_{i,i',j,j',b,b',s,k_s, \tau_{i,j,b} \neq \tau_{i',j',b'}} \tag{32}$$

$$C_{max} \geq C_{i,j,b} \quad \forall_{i,j,b} \tag{33}$$

$$C_{i,j,b} \geq 0 \quad \forall_{i,j,b} \tag{34}$$

$$C_{max} \geq 0 \tag{35}$$

$$\sigma_{i,b} \geq 0 \text{ } e \text{ } inteiro \ \forall_{i,b} \tag{36}$$

$$\delta_{i,b} \in \{0,1\} \tag{37}$$

$$X_{i,j,b,s,k_s} \in \{0,1\} \tag{38}$$

$$\eta_{i,j,b,i',j',b'} \in \{0,1\} \tag{39}$$

A função objetivo representada na equação (23), consiste em minimizar o *makespan*. A equação (24) garante que o somatório do tamanho dos sublotes de um trabalho é superior ou igual à quantidade a produzir desse trabalho. As restrições (25) e (26) identificam os sublotes diferentes de zero e os sublotes vazios aos quais não são atribuídas máquinas. A equação (27) garante que o tamanho dos sublotes seja superior ou igual ao tamanho mínimo de sublote do trabalho correspondente. A restrição (28) assegura que cada sublote com tamanho diferente de zero é atribuído a uma única máquina em cada posto de trabalho. A equação (29) determina o tempo de conclusão de um sublote correspondente à primeira operação de um trabalho. A equação (30) garante que todos os sublotes devem ser processados de acordo com as rotas de

processamento definidas. Deste modo, quando a operação não corresponde à primeira operação de um trabalho considera-se o tempo de conclusão da operação anterior do mesmo trabalho. As restrições (31) e (32) evitam a sobreposição de processamento de dois sublotes atribuídos à mesma máquina. A equação (33) corresponde ao cálculo do *makespan*. Por fim, as restrições (34), (35), (36), (37), (38) e (39) são restrições de sinal.

5.2 Desenvolvimento de um Sistema de Apoio à Decisão

Utilizando as ferramentas Python, a biblioteca PuLP e o *solver* CPLEX, foi desenvolvida uma aplicação que implementa o modelo matemático descrito na secção anterior. Esta ferramenta recebe como entrada (*input*) os dados relativos aos trabalhos a planear e, como saída (*output*), devolve o escalonamento de todos os trabalhos.

A partir da solução obtida para um determinado problema através deste modelo, os trabalhos são escalonados considerando a disponibilidade diária de cada máquina. O fluxograma da Figura 22 descreve o processo aplicado para a realização deste escalonamento. Como se pode observar, o procedimento repete-se até que todos os $T_{i,j,b}$ estejam alocados. Além disso, para cada máquina, é executado uma estrutura de repetição (ciclo *for*) que percorre os vários $T_{i,j,b}$ a serem realizados na mesma.

No fluxograma, "Disp" refere-se ao tempo de disponibilidade diário de cada máquina. O "TR" representa o tempo restante de processamento do $T_{i,j,b}$ em análise, e o "TP" refere-se ao tempo total de processamento necessário para realizar o trabalho, incluindo o tempo de *setup*.

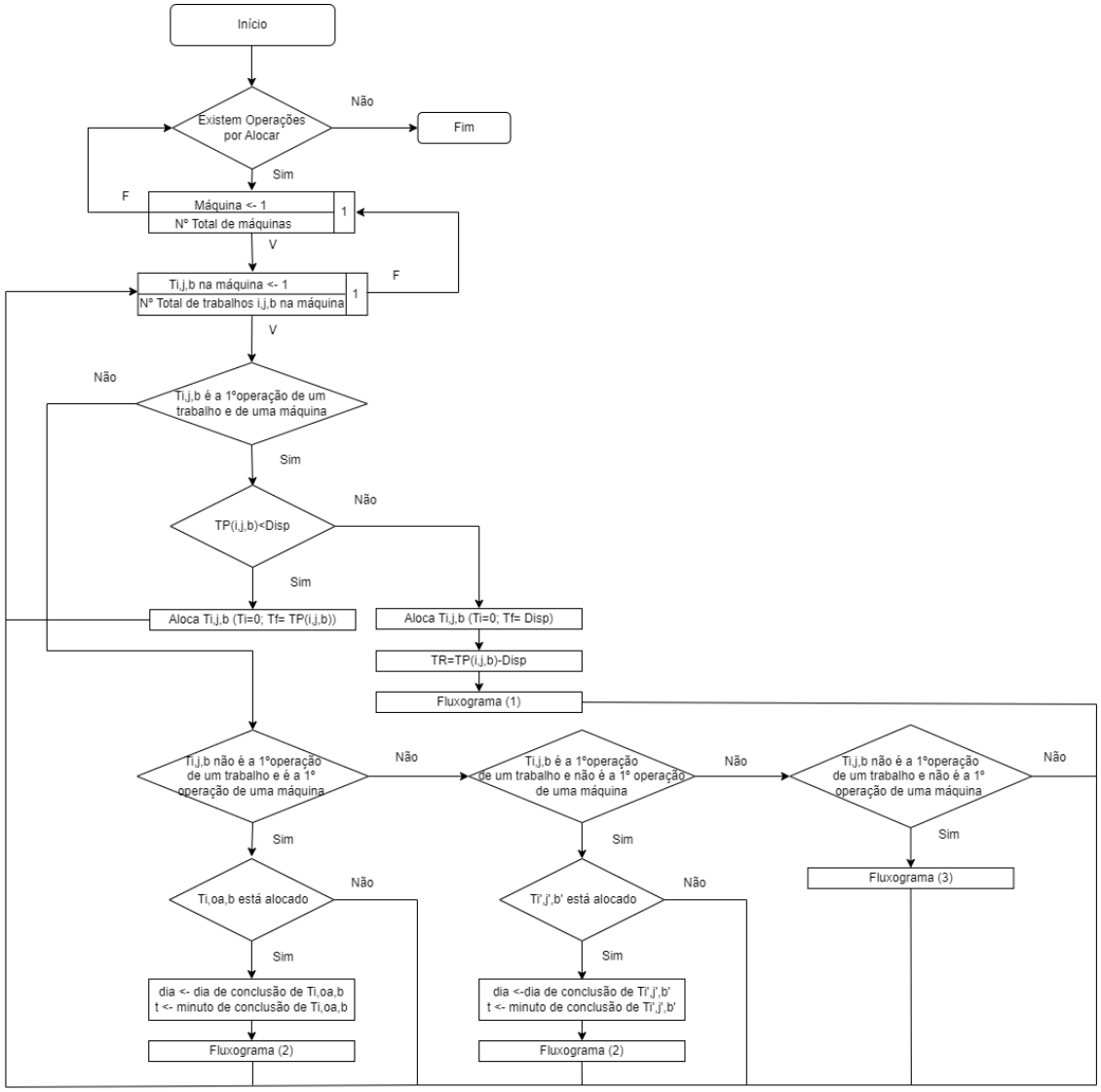


Figura 22 – Fluxograma principal do sistema de apoio à decisão com abordagem *lot streaming*

Na Figura 23, encontra-se o Fluxograma (1) que é utilizado quando o tempo de processamento é superior à disponibilidade da máquina no dia atual e é necessário alocar o trabalho aos dias seguintes até o tempo restante do mesmo ser igual a zero. Quando o $T_{i,j,b}$ estiver totalmente alocado volta ao fluxograma principal da Figura 22.

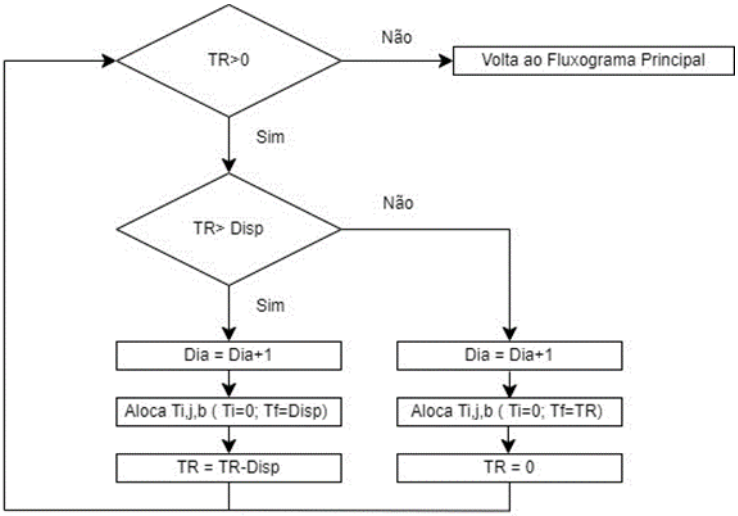


Figura 23 – Fluxograma (1)

O Fluxograma (2) é apresentado na Figura 24, este é introduzido quando se tem de considerar o tempo e o dia de conclusão de uma operação realizada antes da operação em análise.

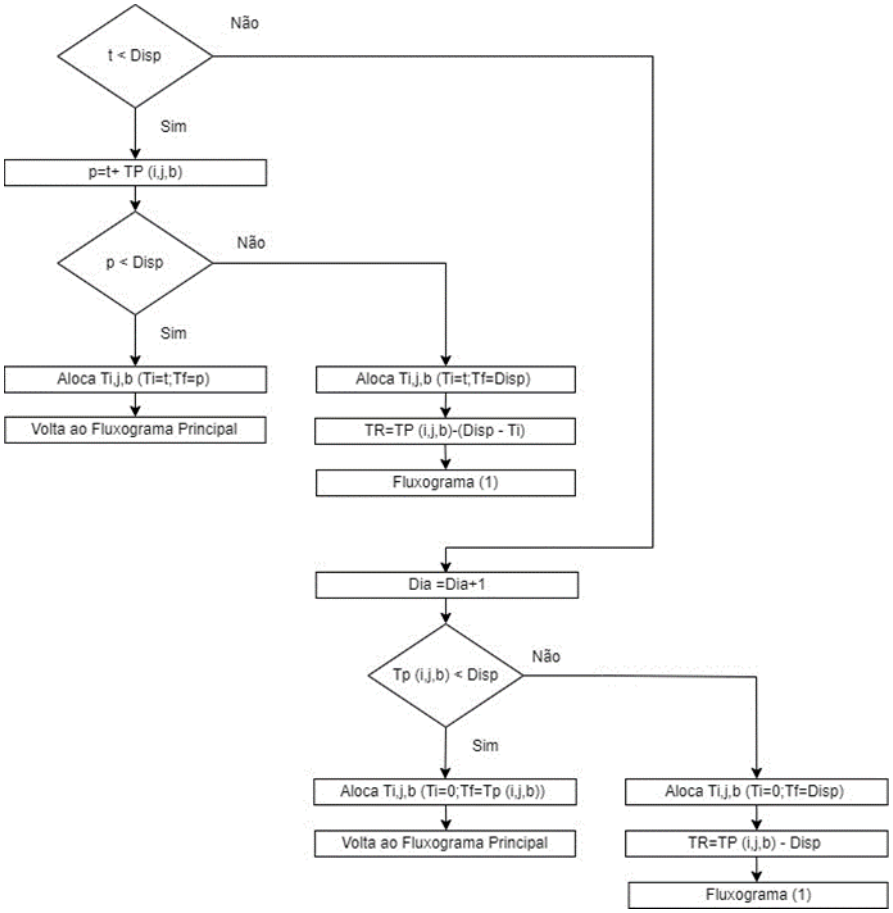


Figura 24 – Fluxograma (2)

Na Figura 25, pode-se visualizar o Fluxograma (3), este é aplicado quando o trabalho em análise não corresponde à primeira operação de um trabalho, nem à primeira operação de uma máquina. Neste caso, é necessário analisar o dia e o tempo de conclusão da operação anterior realizada na mesma máquina e da operação anterior do mesmo trabalho. No caso de os dias serem diferentes é escolhido o dia superior e o tempo de conclusão desse dia. No entanto, se os dias forem iguais escolhe-se o tempo de conclusão superior.

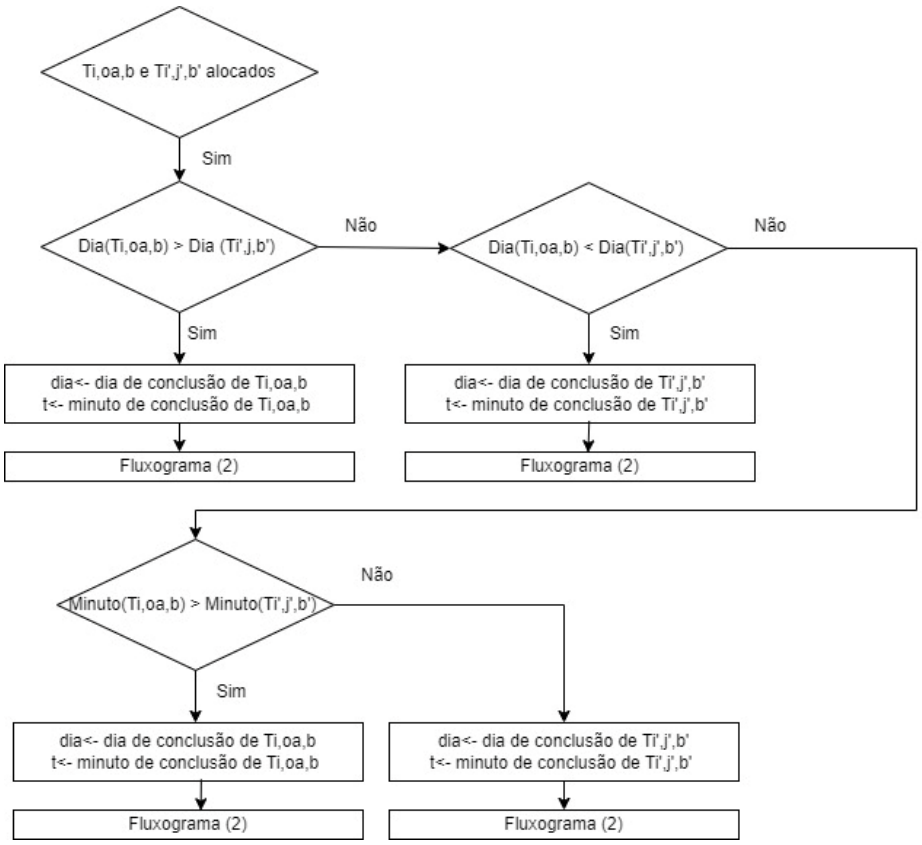


Figura 25 – Fluxograma (3)

Por fim, é criado um gráfico de *Gantt*.

5.3 Experiências Computacionais

Os testes foram realizados num desktop com Intel Core i5 de 2,40 GHz e 8 GB de RAM.

5.3.1 Descrição de uma Instância de Teste e Resultados Obtidos

A seguir, apresenta-se uma instância de teste e os resultados obtidos pelo modelo PLIM2. A instância de teste possui dois trabalhos e as suas respetivas operações, bem como os postos de trabalho onde são realizadas essas operações. A Tabela 25, apresenta os trabalhos e as respetivas operações, que pertencem à instância de teste. Cada trabalho i é composto por um

conjunto de operações, $O_{i,j}$, onde o j representa a ordem da operação. Para cada operação, é especificada a quantidade a ser produzida e o posto de trabalho responsável pela produção.

Tabela 25 – Exemplo de uma instância de teste do modelo PLIM2

Trabalho	Operação	Quantidade	Posto de Trabalho
1	$O_{1,1}$	100	FW
1	$O_{1,2}$	100	DS
2	$O_{2,1}$	90	AW
2	$O_{2,2}$	90	DS

A Tabela 26, apresenta os dados dos postos de trabalho, incluindo a disponibilidade das máquinas, a identificação das máquinas (id), o tempo de *setup* e o tempo de processamento por unidade.

Tabela 26 – Dados dos Postos de Trabalho da instância do modelo PLIM2

Posto de Trabalho	Posto de Trabalho (id)	Disponibilidade (min)	Máquina (id)	*Setup* (min)	Tempo (min)
FW	1	1440	0	5	2
			1	5	2
			2	5	2
DS	11	150	22	5	0,25
AW	2	1440	3	210	1,50
			4	210	1,50
			5	210	1,33

Na Tabela 27, encontram-se os valores relativos a alguns parâmetros, incluindo o número mínimo a produzir por sublote (ϕ_i), o número máximo de sublotes (B_i) e o número de operações que contém um trabalho (N_i).

Tabela 27 – Parâmetros da instância do modelo PLIM2

Trabalho (i)	ϕ_i	B_i	N_i
1	8	12	2
2	9	10	2

De seguida, são apresentados os valores das variáveis de decisão obtidos através do modelo.

Na Tabela 28, encontram-se os valores do tamanho do b-ésimo sublote $(\sigma_{i,b})$, para os quais a variável binária $\delta_{i,b}$ é igual a 1.

Tabela 28 – Resultado das variáveis de decisão $\delta_{i,b}$ e $\sigma_{i,b}$

i	b	$\delta_{i,b}$	$\sigma_{i,b}$
1	10	1	50
1	11	1	50
2	1	1	37
2	2	1	32
2	8	1	21

Observando a Tabela 28, verifica-se que o trabalho 1 foi dividido em dois sublotes iguais, cada um com 50 unidades, e o trabalho 2 foi dividido em 3 sublotes com tamanhos diferentes.

Na Tabela 29, encontram-se as máquinas a que cada sublote foi atribuído, ou seja, os índices para os quais a variável X_{i,j,b,s,k_s} é igual a 1.

Tabela 29 – Resultado da variável de decisão X_{i,j,b,s,k_s}

i	j	b	s	k_s
1	1	10	1	1
1	1	11	1	0
1	2	10	11	22
1	2	11	11	22
2	1	1	2	3
2	1	2	2	5
2	1	8	2	4
2	2	1	11	22
2	2	2	11	22
2	2	8	11	22

Na Figura 26, está representada graficamente a solução obtida com o modelo.

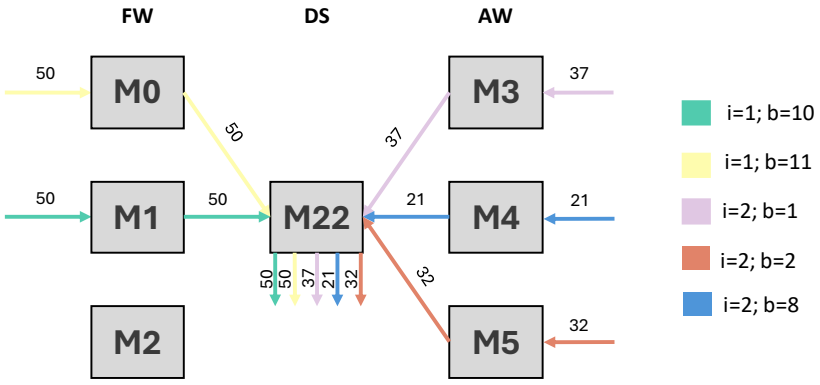


Figura 26 – Representação gráfica da solução encontrada com o modelo PLIM2

Conforme ilustrado na Tabela 29, vários sublotes são processados na máquina 22. Para ordenar esses sublotes na mesma máquina, é necessário verificar os valores da variável $\eta_{i,j,b,i',j',b'}$. Quando o seu valor é igual a 1, significa que i, j, b é realizado depois de i', j', b'. Os valores dos índices da variável $\eta_{i,j,b,i',j',b'}$ quando esta tem o valor de 1, estão apresentados na Tabela 30.

Tabela 30 – Valor da variável de decisão $\eta_{i,j,b,i',j',b'}$

i	j	b	i'	j'	b'
1	2	11	1	2	10
2	2	1	1	2	10
2	2	2	1	2	10
2	2	8	1	2	10
2	2	1	1	2	11
2	2	2	1	2	11
2	2	8	1	2	11
2	2	1	2	2	2
2	2	1	2	2	8
2	2	2	2	2	8

A Tabela 31 mostra a sequência dos trabalhos $T_{i,j,b}$ para cada máquina, indicando a ordem em que estes devem ser processados.

Tabela 31 – Resultado da sequência de trabalhos obtida pelo modelo PLIM2

Máquina (id)	**Sequência de Trabalhos ($T_{i,j,b}$)**
0	$T_{1,1,11}$
1	$T_{1,1,10}$
2	$-$
22	$T_{1,2,10} \rightarrow T_{1,2,11} \rightarrow T_{2,2,8} \rightarrow T_{2,2,2} \rightarrow T_{2,2,1}$
3	$T_{2,1,1}$
4	$T_{2,1,8}$
5	$T_{2,1,2}$

Na Tabela 32, são apresentados os tempos de conclusão. A variável $C_{i,j,b}$ corresponde ao tempo de conclusão do sublote b, da operação j e do trabalho i.

Tabela 32 – Resultado da variável de decisão $C_{i,j,b}$

i	j	b	$C_{i,j,b}$
1	1	10	105
1	1	11	105
1	2	10	122,50
1	2	11	140
2	1	1	265,50
2	1	2	252,67
2	1	8	241,50
2	2	1	279,92
2	2	2	265,67
2	2	8	251,75

O valor do *makespan* é igual ao maior valor de todos os tempos de conclusão ($C_{i,j,b}$). Neste caso, corresponde a 279,92 minutos.

O tempo CPU utilizado para resolver este problema foi de 0,0625 segundos.

O cálculo do tempo de conclusão, segue um processo específico. Primeiramente, é determinado o "Tempo inicial". Para isso é considerado para o $T_{i,j,b}$ o tempo de conclusão da operação anterior ($T_{i,j',b}$) e o tempo de conclusão do $T_{i,j,b}$ realizado na posição anterior da mesma máquina. O maior desses tempos é o escolhido para o "Tempo Inicial". Com base nisso, o tempo de conclusão de $T_{i,j,b}$ é determinado pela fórmula apresentada na equação (40):

$$\boldsymbol{C_{i,j,b} = Tempo\ Inicial + \sigma_{i,b} * P_{s,k_s} + S_{s,k_s}} \quad (40)$$

De seguida, na Tabela 33, são apresentados os cálculos realizados para validar os valores obtidos.

Tabela 33 - Validação variável $C_{i,j,b}$

i	j	b	$C_{i,j,b}$
1	1	10	50×2+5=105
1	1	11	50×2+5=105
1	2	10	105+50×0,25+5=122,50
1	2	11	122,50+50×0,25+5=140
2	1	1	37×1,50 +210=265,50
2	1	2	32×1,33+210=252,67
2	1	8	21×1,50 +210=241,50
2	2	1	265,67+37×0,25+5=279,92
2	2	2	252,67+32×0,25+5=265,67
2	2	8	241,50 +21×0,25+5=251,75

Analisando a Tabela 33, verifica-se que os valores obtidos pelo modelo estão corretos. Em seguida, a Tabela 34 apresenta o escalonamento obtido, considerando a restrição de disponibilidade diária das máquinas.

Tabela 34 – Escalonamento da solução obtida pelo modelo PLIM2

Máquina	**Dia**	**Tempo Inicial**	**Tempo Final**	$T_{i,j,b}$
1	11/01/2023	0	105	$T_{1,1,10}$
0	11/01/2023	0	105	$T_{1,1,11}$
22	11/01/2023	106	123,50	$T_{1,2,10}$
22	11/01/2023	124,5	142	$T_{1,2,11}$
3	11/01/2023	0	265,50	$T_{2,1,1}$
5	11/01/2023	0	252,67	$T_{2,1,2}$
4	11/01/2023	0	241,50	$T_{2,1,8}$
22	12/01/2023	0	10,25	$T_{2,2,8}$
22	12/01/2023	11,25	24,25	$T_{2,2,2}$
22	12/01/2023	25,25	39,50	$T_{2,2,1}$

A disponibilidade diária da máquina 22 é de 150 minutos. Assim, as operações $O_{2,2}$ realizadas nesta máquina, tiveram de ser escalonadas para o dia seguinte, uma vez que as operações $O_{2,1}$

excedem o limite de 150 minutos. A data inicial considerada é fornecida nos dados como data de lançamento dos trabalhos, que neste caso, é 11/01/2023.

Com base nos dados da Tabela 34, foi criado um gráfico de *Gantt* para cada dia, conforme apresentado na Figura 27. No gráfico, o eixo das abcissas representa o tempo em minutos e o eixo das ordenadas os índices das máquinas. Além disso, a legenda do gráfico utiliza cores para representar os diferentes trabalhos i e os respetivos sublotes b.

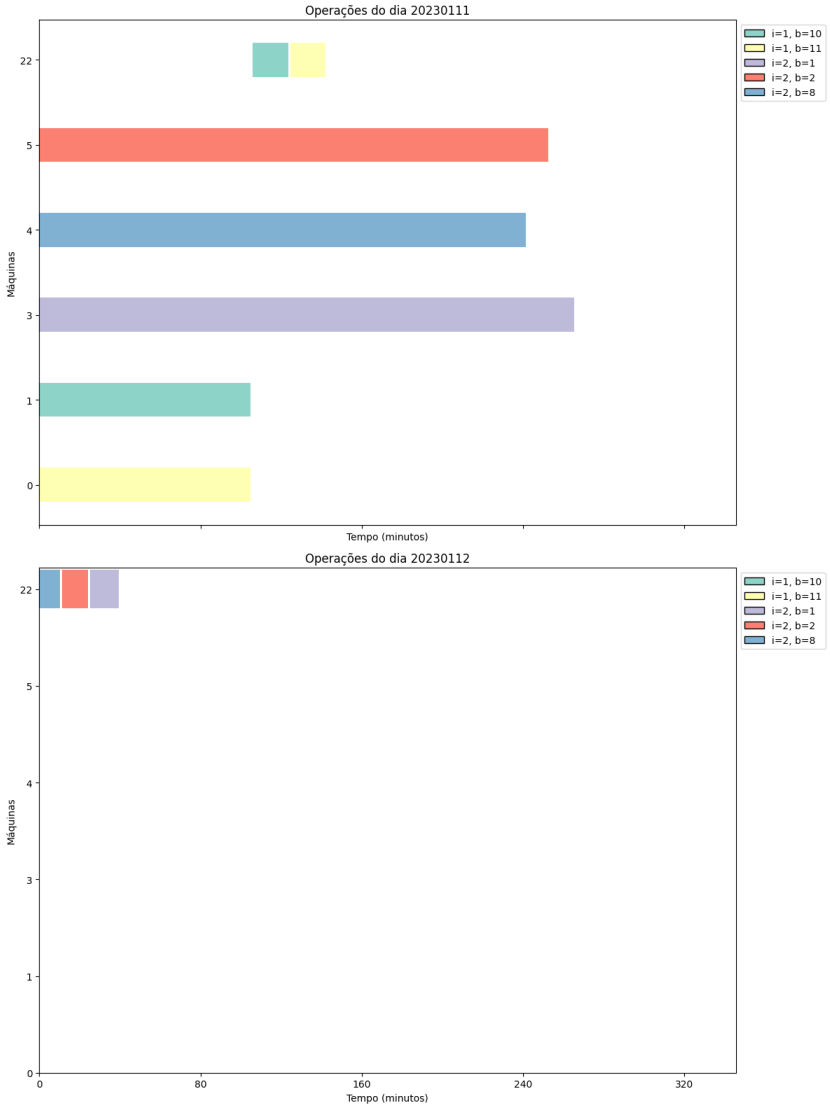


Figura 27 – Gráfico de *Gantt* da solução obtida para a instância de teste com o modelo PLIM2

5.3.2 Análise dos Tempos Computacionais em Função da Variação do Número de Trabalhos e Quantidade nas Instâncias de Teste no Modelo PLIM2

Na presente secção, analisam-se os tempos computacionais para diferentes instâncias de teste. O objetivo foi avaliar o desempenho do modelo PLIM2 em relação ao aumento da complexidade do problema. Os tempos computacionais são registados para cada configuração, com o intuito de analisar como o aumento da complexidade do problema afeta o tempo de execução para resolver o PLIM2.

Inicialmente, foram realizados testes mantendo o número de operações por trabalho constante, igual a dois e variando o número de trabalhos, bem como a quantidade a produzir. Na Tabela 35, são apresentados os respetivos valores do tempo de CPU obtidos. Nas instâncias com 4 trabalhos e quantidade igual a 70 ou mais unidades não foi possível obter nenhuma solução em tempo razoável.

Tabela 35 – Tempos CPU variando o número de trabalhos e a quantidade a produzir

Quantidade a produzir	2	3	4
50	0,19	0,08	0,09
70	0,02	0,31	-
90	0,11	0,55	-
110	0,20	0,69	-

Na Figura 28, encontra-se um gráfico ilustrativo dos valores apresentados na Tabela 35.

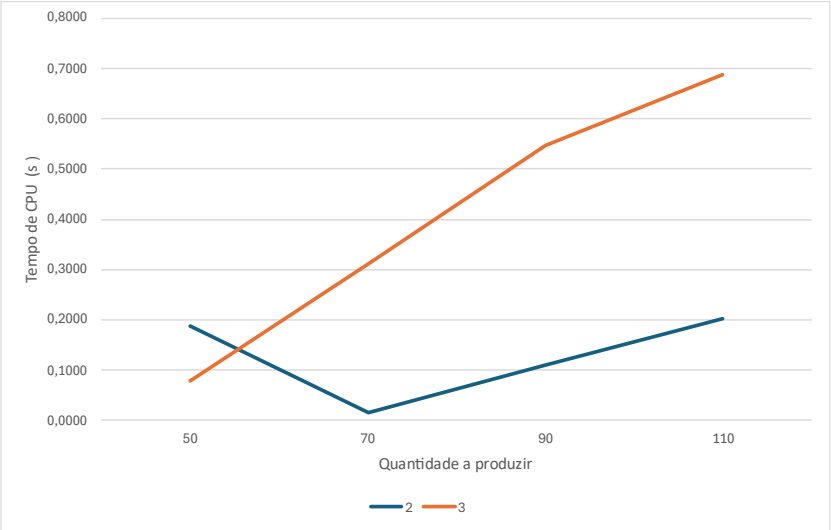


Figura 28 - Tempos de CPU variando o número de trabalhos e a quantidade

Através do gráfico da Figura 28, verifica-se que o aumento do número de trabalhos e da quantidade a produzir, ou seja, da complexidade do problema, produz impacto no tempo de CPU.

5.3.3 Análise dos Tempos Computacionais em Função da Variação do Número de Operações e Quantidade nas Instâncias de Teste no Modelo PLIM2

Nesta secção analisam-se os tempos computacionais em relação à variação do número de operações e da quantidade. Neste caso, o número de trabalhos é constante e igual a 2. Na Tabela 36, são apresentados os valores obtidos.

Tabela 36 - Tempos CPU variando o número de operações e a quantidade a produzir

Quantidade a produzir	2	3	4	5
50	0,13	0,05	0,09	0,17
70	0,05	0,20	0,17	0,09
90	0,14	0,17	0,22	0,23
110	0,23	0,23	0,28	0,39

Na Figura 29 são representados graficamente os valores da Tabela 36 para uma melhor compreensão dos valores.

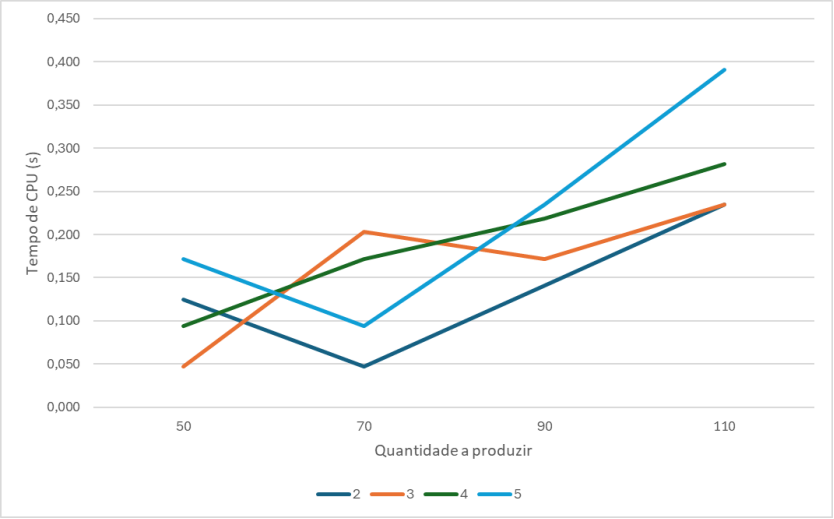


Figura 29 - Tempos de CPU variando o número de operações e a quantidade a produzir

5.3.4 Validação das Soluções

Neste processo, em que foram realizados vários testes, durante a validação das soluções verificou-se que existiam tempos de conclusão não expectáveis. Pelo que suspeitou-se que a origem do erro poderia estar no valor do Ω utilizado. Já que observou-se que este tinha interferência na determinação da solução.

Assim, optou-se por seguir uma abordagem, em que para cada conjunto de restrições do tipo $Z_j \geq X_j - \left(1 - Y_{i,j}\right) * M_j$, o M_j deve ser o limite superior de X_j. Desta forma, foram definidos três Ω diferentes.

O Ω da restrição (25) corresponde ao maior valor de entre todas as quantidades a produzir de todos os trabalhos, tal como mostra a equação (41).

$$\Omega\mathbf{1} = \mathbf{max}(\boldsymbol{E_i}) \ \forall \boldsymbol{i} \tag{41}$$

O Ω da restrição (29) é calculado do modo apresentado na equação (42), ou seja, para a primeira operação de cada trabalho é calculado o tempo de processamento e no final é escolhido o maior valor para o Ω.

$$\Omega\mathbf{2} = \mathbf{max}\ (\boldsymbol{E_i} * \boldsymbol{P_{s,k_s}} + \boldsymbol{S_{s,k_s}}) \ \ \forall_{\boldsymbol{i,j=1,s,k_s}} \tag{42}$$

O Ω das restantes restrições é calculado da forma demonstrada na equação (43), ou seja, é o somatório da soma do tempo de processamento para cada trabalho e operação, para cada uma das máquinas disponíveis no posto de trabalho dessa operação.

$$\Omega\mathbf{3} = \sum_{\boldsymbol{i}}^{\boldsymbol{n}} \sum_{\boldsymbol{j}}^{\boldsymbol{N_i}} \mathbf{max}\big(\boldsymbol{E_i} * \boldsymbol{P_{s,k_s}} + \boldsymbol{S_{s,k_s}} \ \ \forall_{\boldsymbol{s,k_s}}\ \big) \tag{43}$$

Com esta abordagem, foram realizados vários testes e constatou-se que para problemas de grande dimensão o problema mantinha-se.

Deste modo, não foi possível encontrar soluções viáveis para problemas de grandes dimensões aplicando o modelo apresentado. Esta dificuldade prende-se ao crescimento exponencial da complexidade computacional relacionado com o aumento do tamanho das instâncias e na dificuldade em coordenar o processamento de múltiplos sub-lotes em várias máquinas.

5.4 Comparação dos Resultados Obtidos pelos Modelos PLIM1 e PLIM2

Neste capítulo, comparam-se os resultados obtidos pelo modelo PLIM1 utilizando o *job-splitting* e pelo PLIM2 usando o *lot streaming* (Tabela 37).

Tabela 37 – Comparação entre o PLIM1 utilizando o *job-splitting* e o PLIM2 usando o *lot streaming*

Instância	**Nº Trabalhos**	**Nº Operações**	**Quantidade a produzir**	***Makespan* (min) PLIM1**	***Makespan* (min) PLIM2**
1	2	2	110	295,83	293
2	2	3	110	320,36	305,81
3	2	4	110	347,36	319,61
4	2	5	110	376,86	332,61

Com base na Tabela 37, pode-se concluir que os resultados obtidos com a técnica de *lot streaming* são superiores aos da técnica *job-splitting* em termos de *makespan*. No entanto, os tempos computacionais do *lot streaming* são mais elevados.

6 Conclusão

Neste capítulo são apresentadas as conclusões finais do trabalho realizado, bem como o trabalho futuro.

6.1 Conclusões Finais

O presente trabalho foi desenvolvido com base num problema real da indústria têxtil, abordando o escalonamento em sistemas *Job-Shop* flexíveis com divisão de lotes.

Inicialmente, foi proposto um modelo de programação linear inteira mista com o objetivo de minimizar o tempo de conclusão de todos os trabalhos. Utilizou-se a técnica de *job-splitting*, que permite a divisão de um lote em sublotes em operações com várias máquinas disponíveis, sendo necessário aguardar a finalização de todos os sublotes para que o trabalho possa prosseguir. Este modelo foi implementado em Python com o auxílio do CPLEX, e apresentou resultados viáveis para instâncias pequenas, porém, para instâncias de tamanho médio, revelou-se incapaz de encontrar soluções num tempo computacional aceitável.

Para superar esta limitação e possibilitar a resolução de instâncias reais, foi desenvolvido um Algoritmo Genético Híbrido, ao qual foi incorporado um Algoritmo de Pesquisa Local para otimizar o número e o tamanho dos sublotes. A meta-heurística foi implementada em Python e testada com instâncias de diferentes dimensões. Os testes realizados mostram que o tempo de CPU é afetado significativamente, com a dimensão do problema. Além disso, foram realizadas avaliações em três instâncias reais da indústria têxtil. Para uma instância média com 57 trabalhos, o tempo de CPU foi de aproximadamente 20 minutos, enquanto para uma instância com 67 trabalhos, o tempo foi de 35 minutos. Estes resultados são promissores e indicam que a meta-heurística pode ser aplicada na indústria, oferecendo uma solução eficiente para o processo de escalonamento de trabalhos.

A meta-heurística apresentada permitiu alcançar o objetivo principal deste trabalho, que era resolver o problema *Job-Shop* flexível com divisão de lotes em instâncias reais. No entanto, também foi explorada uma outra técnica de divisão de lotes, o *lot streaming*, na qual o trabalho é dividido em sublotes no início e estes são tratados de forma independente ao longo do processo produtivo. Foi implementado um modelo de programação linear inteira mista utilizando esta técnica, e os resultados indicam que os valores de *makespan* obtidos com esse método são superiores aos do *job-splitting*. Contudo, esse modelo não se mostrou viável para todas as instâncias e apresentou tempos computacionais mais elevados.

Este trabalho reforça a importância das técnicas desenvolvidas e testadas, e mostra que a aplicação de meta-heurísticas pode facilitar a obtenção de soluções em problemas de escalonamento reais, em particular, da indústria têxtil.

6.2 Trabalho Futuro

Como trabalho futuro será adaptada a meta-heurística apresentada para a técnica *lot streaming* e comparados os resultados obtidos. Esta comparação permitirá avaliar a proximidade das soluções geradas em relação à solução ótima, além de comparar os tempos de CPU, proporcionando uma análise mais completa sobre qual técnica de divisão de lotes é mais eficiente.

Referências

Blackburn, J. D. 1991. *Time-based competition: the next battleground in American manufacturing*. Homewood: Business One Irwin.

Bockerstette, J. A. & Shell, R. L. 1993. *Time based manufacturing*. EUA: McGraw-Hill Inc.

Boyer, V., Vallikavungal, J., Cantú Rodríguez, X., & Salazar-Aguilar, M. A. 2021. The generalized flexible job shop scheduling problem. *Computers & Industrial Engineering*, 160, 107542. Available from: https://doi.org/10.1016/j.cie.2021.107542

Bożek, A., & Werner, F. 2018. Flexible job shop scheduling with lot streaming and sublot size optimisation. *International Journal of Production Research*, 56(19), 6391–6411. Available from: https://doi.org/10.1080/00207543.2017.1346322

Cavallaro, C., Cutello, V., Pavone, M., & Zito, F. 2024. Machine Learning and Genetic Algorithms: A case study on image reconstruction. *Knowledge-Based Systems*, 284, 111194. Available from: https://doi.org/10.1016/j.knosys.2023.111194

Chang, J. H., & Chiu, H. N. 2005. A comprehensive review of lot streaming. *International Journal of Production Research*, 43(8), 1515–1536. Available from: https://doi.org/10.1080/00207540412331325396

Chaudhry, I. A., & Khan, A. A. 2016. A research survey: Review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3), 551–591. Available from: https://doi.org/10.1111/itor.12199

Chen, J., & Steiner, G. 1997. Approximation methods for discrete lot streaming in flow shops. *Operations Research Letters*, 21(3), 139–145. Available from: https://doi.org/10.1016/S0167-6377(97)00039-4

Cheng, M., Mukherjee, N. J., & Sarin, S. C. 2013. A review of lot streaming. *International Journal of Production Research*, 51(23–24), 7023–7046. Available from: https://doi.org/10.1080/00207543.2013.774506

Da Costa, L. A. A. F. 2003. *Algoritmos Evolucionários em Optimização Uni e Multi-objectivo*. Doctoral dissertation, Universidade do Minho.

Dauzère-Pérès, S., Ding, J., Shen, L., & Tamssaouet, K. 2024. The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2), 409–432. Available from: https://doi.org/10.1016/j.ejor.2023.05.017

Defersha, F. M., & Bayat Movahed, S. 2018. Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming. *Computers & Industrial Engineering*, 117, 319–335. Available from: https://doi.org/10.1016/j.cie.2018.02.010

Defersha, F. M., Obimuyiwa, D., & Yimer, A. D. 2022. Mathematical model and simulated annealing algorithm for setup operator constrained flexible job shop scheduling problem. *Computers & Industrial Engineering*, 171, 108487. Available from: https://doi.org/10.1016/j.cie.2022.108487

Demir, Y., & İşleyen, S. K. 2014. An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations. *International Journal of Production Research*, 52(13), 3905–3921. Available from: https://doi.org/10.1080/00207543.2014.889328

Dueck, G., & Scheuer, T. 1990. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1), 161–175. Available from: https://doi.org/10.1016/0021-9991(90)90201-B

Fan, J., Zhang, C., Shen, W., & Gao, L. 2023. A matheuristic for flexible job shop scheduling problem with lot-streaming and machine reconfigurations. *International Journal of Production Research*, 61(19), 6565–6588. Available from: https://doi.org/10.1080/00207543.2022.2135629

Gao, K. Z., Suganthan, P. N., Pan, Q. K., Chua, T. J., Cai, T. X., & Chong, C. S. 2016. Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. *Journal of Intelligent Manufacturing*, 27(2), 363–374. Available from: https://doi.org/10.1007/s10845-014-0869-8

Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5), 533–549. Available from: https://doi.org/10.1016/0305-0548(86)90048-1

Gmys, J., Mezmaz, M., Melab, N., & Tuyttens, D. 2020. A computationally efficient Branch-and-Bound algorithm for the permutation flow-shop scheduling problem. *European Journal of Operational Research*, 284(3), 814–833. Available from: https://doi.org/10.1016/j.ejor.2020.01.039

Goldberg, D. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. AddisonWesley.

Harjunkoski, I., Maravelias, C. T., Bongers, P., Castro, P. M., Engell, S., Grossmann, I. E., Hooker, J., Méndez, C., Sand, G., & Wassick, J. 2014. Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, 62, 161–193. Available from: https://doi.org/10.1016/j.compchemeng.2013.12.001

Holland, J. H. 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

Jeong, H.-I., Park, J., & Leachman, R. C. 1999. A batch splitting method for a job shop scheduling problem in an MRP environment. *International Journal of Production Research*, 37(15), 3583–3598. Available from: https://doi.org/10.1080/002075499190194

Kennedy, J., & Eberhart, R. 1995. Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4, 1942–1948 vol.4. Available from: https://doi.org/10.1109/ICNN.1995.488968

Kirkpatrick, S., Gelatt, C., & Vecchi, M. 1983. Optimization by Simulated Annealing. *Science*, 220(4598), 671–680. Available from: https://doi.org/10.1126/science.220.4598.671

Kuruoglu, E. E., Kuo, C. L., & Chan, W. K. V. 2023. Sparse neural network optimization by Simulated Annealing. *Franklin Open*, 4, 100037. Available from: https://doi.org/10.1016/j.fraope.2023.100037

Ku, W.-Y., & Beck, J. C. 2016. Mixed Integer Programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73, 165–173. Available from: https://doi.org/10.1016/j.cor.2016.04.006

Land, A. H., & Doig, A. G. 1960. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3), 497. Available from: https://doi.org/10.2307/1910129

Liangxiao, J., & Zhongjun, D. 2015. An Improved Genetic Algorithm for Flexible Job Shop Scheduling Problem. *2015 2nd International Conference on Information Science and Control Engineering*, 127–131. Available from: https://doi.org/10.1109/ICISCE.2015.36

Li, X., & Gao, L. 2016. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 174, 93–110. Available from: https://doi.org/10.1016/j.ijpe.2016.01.016

Meng, T., Pan, Q.-K., & Sang, H.-Y. 2018. A hybrid artificial bee colony algorithm for a flexible job shop scheduling problem with overlapping in operations. *International Journal of Production Research*, 56(16), 5278–5292. Available from: https://doi.org/10.1080/00207543.2018.1467575

Mladenović, N., & Hansen, P. 1997. Variable neighborhood search. *Computers & operations research*, 24(11), 1097-1100.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., & Sewell, E. C. 2016. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19, 79–102. Available from: https://doi.org/10.1016/j.disopt.2016.01.005

Muthuraman, S., & Venkatesan, V. P. 2017. A Comprehensive Study on Hybrid Meta-Heuristic Approaches Used for Solving Combinatorial Optimization Problems. *2017 World Congress on Computing and Communication Technologies (WCCCT)*, 185–190. Available from: https://doi.org/10.1109/WCCCT.2016.53

Naderi, B., Ruiz, R., & Roshanaei, V. 2023. Mixed-Integer Programming vs. Constraint Programming for Shop Scheduling Problems: New Results and Outlook. *INFORMS Journal on Computing*, 35(4), 817–843. Available from: https://doi.org/10.1287/ijoc.2023.1287

Park, M.-J., & Ham, A. 2022. Energy-aware flexible job shop scheduling under time-of-use pricing. *International Journal of Production Economics*, 248, 108507. Available from: https://doi.org/10.1016/j.ijpe.2022.108507

Pezzella, F., Morganti, G., & Ciaschetti, G. 2008. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers & Operations Research*, 35(10), 3202–3212. Available from: https://doi.org/10.1016/j.cor.2007.02.014

Pinedo, M., & Chao, X. 1999. *Operations scheduling with applications in manufacturing and services*.New York: McGraw-Hill.

Pinedo, M. L. 2012. Scheduling Theory, Algorithms and Systems (5th ed.). *Springer*. Available from: https://doi.org/10.1007/978-3-319-26580-3

Potts, C. N., & Baker, K. R. 1989. Flow shop scheduling with lot streaming. *Operations Research Letters*, 8(6), 297–303. Available from: https://doi.org/10.1016/0167-6377(89)90013-8

Reiter, S. 1966. A System for Managing Job-Shop Production. *The Journal of Business*, 39(3), 371–393.

Shen, L., Dauzère-Pérès, S., & Neufeld, J. S. 2018. Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2), 503–516. Available from: https://doi.org/10.1016/j.ejor.2017.08.021

Shojaee, M., Noori, S., Jafarian-Namin, S., & Johannssen, A. 2024. Integration of production–maintenance planning and monitoring simple linear profiles via Hotelling's T2 control chart and particle swarm optimization. *Computers & Industrial Engineering*, 188, 109864. Available from: https://doi.org/10.1016/j.cie.2023.109864

Singh, M. R., & Mahapatra, S. S. 2016. A quantum behaved particle swarm optimization for flexible job shop scheduling. *Computers & Industrial Engineering*, 93, 36–44. Available from: https://doi.org/10.1016/j.cie.2015.12.004

Tutumlu, B., & Saraç, T. 2023. A MIP model and a hybrid genetic algorithm for flexible job-shop scheduling problem with job-splitting. *Computers & Operations Research*, 155, 106222. Available from: https://doi.org/10.1016/j.cor.2023.106222

Wang, L., Hu, X., Wang, Y., Xu, S., Ma, S., Yang, K., Liu, Z., & Wang, W. 2021. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Computer Networks*, 190, 107969. Available from: https://doi.org/10.1016/j.comnet.2021.107969

Zhang, J., Ding, G., Zou, Y., Qin, S., & Fu, J. 2019. Review of job shop scheduling research and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing*, 30(4), 1809–1830. Available from: https://doi.org/10.1007/s10845-017-1350-2

Zhang, J., & Yang, J. 2016. Flexible job-shop scheduling with flexible workdays, preemption, overlapping in operations and satisfaction criteria: An industrial application. *International Journal of Production Research*, 54(16), 4894–4918. Available from: https://doi.org/10.1080/00207543.2015.1134839

Zhang, Y., Zhang, Z., Zeng, Y., & Wu, T. 2023. Constraint programming for multi-line parallel partial disassembly line balancing problem with optional common stations. *Applied Mathematical Modelling*, 122, 435–455. Available from: https://doi.org/10.1016/j.apm.2023.06.009

Anexo 1 – Declaração de Integridade

DECLARAÇÃO DE INTEGRIDADE

Declaro ter conduzido este trabalho académico com integridade. Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.
Declaro que o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.
Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 15 de setembro de 2024

Apêndice A – Resultado dos Testes para Determinar o Tamanho da População

Tabela 38 – Resultado do Testes para Determinar o Tamanho da População

	Instância 1				Instância 2			
Tam. População	100	150	200	250	100	150	200	250
CPU 1	0,58	0,36	0,73	1,02	0,48	1,27	2,2	1,78
Makespan 1	478	514,33	458,50	453,50	684,33	708,33	684,33	703,33
CPU 2	0,50	0,781	0,88	0,83	0,50	1,45	1,94	2,63
Makespan 2	485,50	478,50	463	469	688,67	683,33	710,50	707,5
CPU 3	0,64	0,58	0,47	1,03	1,16	1,41	1,98	2,41
Makespan 3	478	488	471,50	441,8	701,67	691,83	684,33	703,33
CPU 4	0,563	0,66	0,91	1,11	0,98	1,08	1,97	2,20
Makespan 4	468,33	513	475	464,50	710,50	710,50	706	701,17
CPU 5	0,56	0,63	1,05	1,08	0,77	1,55	2,27	2,53
Makespan 5	508	498	468	459,50	710,5	703,83	710,50	683,33
CPU 6	0,36	0,48	0,50	1,13	0,91	1,61	1,84	2,72
Makespan 6	498	463	458,50	473,83	688,67	693	683,33	693
CPU 7	0,531	0,58	0,75	0,7	1,06	1,64	1,98	3,08
Makespan 7	491,50	468	472	479,17	684,33	693,33	710,50	688,33
CPU 8	0,516	0,58	0,72	1,05	1,25	1,36	2,42	3,05
Makespan 8	485,50	440,50	469,50	464,50	693,	698,33	703,50	710,50
CPU 9	0,45	0,73	0,66	1,08	1,16	1,56	1,36	2,89
Makespan 9	485,50	507,33	478	471,50	697,33	684,33	688,33	688,33
CPU 10	0,56	0,72	1,17	1,2	1,11	1,70	1,80	2,36
Makespan 10	494,33	497,50	479,50	490,50	698,33	693	684,33	697,33
CPU Média	0,53	0,61	0,78	1,04	0,94	1,46	1,98	2,56
Makespan Média	487,27	486,82	469,35	466,78	695,73	695,98	696,57	697,62

Apêndice B – Resultados dos Testes para Determinar a Taxa de Cruzamento

Tabela 39 – Testes para a determinação da taxa de cruzamento

	Instância 1					Instância 2				
Taxa Cruzamento	0,5	0,6	0,7	0,8	0,9	0,5	0,6	0,7	0,8	0,9
CPU 1	1,52	0,38	0,41	0,64	0,38	0,34	0,55	0,67	0,86	0,81
Makespan 1	515,67	489,83	481,33	473,50	492	710,50	738,33	684,33	714,67	710,50
CPU 2	0,36	0,63	0,47	0,53	0,48	0,42	0,61	0,78	0,64	0,86
Makespan 2	516,50	489,50	491,50	464,00	508	720,50	705,17	710,50	684,33	710,50
CPU 3	0,31	0,38	0,42	0,31	0,33	0,59	0,81	1,00	0,66	0,89
Makespan 3	459,50	493,83	519	485,50	504	714,67	693	710,50	683,33	710,50
CPU 4	0,42	0,06	0,58	0,48	0,48	0,44	0,63	0,81	0,64	0,64
Makespan 4	483	494,33	508	508,00	537,67	727,67	698,33	706,00	707,50	688,33
CPU 5	0,40	0,44	0,42	0,48	0,41	0,58	1,03	0,64	0,81	0,75
Makespan 5	523	508	520,33	493,50	517,50	710,50	718,33	688,33	688,33	710,50
CPU 6	0,44	0,66	0,58	0,63	0,66	0,66	1,14	0,75	0,52	0,67
Makespan 6	488	467,50	481	508,00	493	699,83	703	720,50	719,00	707,50
CPU 7	0,45	0,47	0,52	0,61	0,63	0,77	0,88	0,69	0,41	0,83
Makespan 7	516	508	513	489,50	519	710,50	713,33	723,33	684,33	695
CPU 8	0,27	0,53	0,47	0,58	0,39	0,59	0,39	1,16	0,91	0,84
Makespan 8	517	497,50	513	483,50	517	688,33	720,50	713,33	710,50	697,17
CPU 9	0,59	0,45	0,41	0,47	0,53	0,83	0,69	0,75	0,89	0,67
Makespan 9	485,50	481,33	514	509,50	471,50	717,33	719,83	713,33	703,00	720,50
CPU 10	0,73	0,63	0,44	0,48	0,58	0,59	0,81	0,48	0,78	0,80

Makespan 10	496,50	494,33	508	481,33	470,50	698,33	710,50	683,33	719	688,33
CPU Média	0,55	0,46	0,47	0,52	0,49	0,58	0,75	0,77	0,71	0,78
Makespan Média	500,07	492,42	504,92	489,63	503,02	709,82	712,03	705,35	701,40	703,88

Apêndice C – Resultados dos Testes para Determinar do Valor do Parâmetro β

Tabela 40 - Testes para a determinação do valor de β

	Instância 1					Instância 2				
β	10	20	30	40	50	10	20	30	40	50
CPU 1	1,86	2,08	1,56	1,31	1,05	3,25	2,38	2,69	2,95	1,72
Makespan 1	543	499,50	508	489,50	449,50	710,50	710,50	671,33	710,33	688,33
CPU 2	3	1,94	1,14	1,39	1,47	3,02	2,83	2,86	2,42	1,61
Makespan 2	515,67	443	521,50	489,50	458,50	706	732	723,33	728,33	683,33
CPU 3	2,422	1,55	1,53	1,41	1,33	2,73	2,06	3,06	2,30	2,25
Makespan 3	501,50	428,50	508	538	519	728,30	688,33	723,33	720,50	694,50
CPU 4	2,16	1,28	1,52	1,55	1,45	2,80	1,92	3,02	2,08	2,20
Makespan 4	520	493	451	458,50	473,50	733,30	710,50	710,50	710,50	697,33
CPU 5	1,58	1,55	1,69	1,39	1,14	2,36	2,05	1,94	1,98	1,69
Makespan 5	547,50	532,50	453,83	549	470,50	738,30	671,50	723,33	713,33	710,33
CPU 6	1,80	1,84	1,45	1,39	1,52	3,34	3,53	2,84	1,97	1,63
Makespan 6	498	519	508	503,50	508	728,30	723,33	710,50	733,33	713,33
CPU 7	2	1,55	1,73	0,95	1,22	2,22	3,34	3,91	2,44	1,88
Makespan 7	507,33	523	511,50	477,83	489,90	720,50	708,33	710,50	680	710,50
CPU 8	1,25	1,59	1,66	1,23	0,81	3	3,09	1,77	1,59	2,02
Makespan 8	525,50	483	469	488	459,20	728,33	710,50	680	703	688,33
CPU 9	1,69	2,13	1,58	1,44	1,20	2,88	2,67	2,20	2,03	1,69
Makespan 9	493	464	508	464	483	713,33	707,50	671,50	710,33	710,50
CPU 10	2,86	1,70	1,55	1,06	1,33	3,20	2,47	1,88	2,48	1,78
Makespan 10	516	498	508,50	508	463,50	736,30	710,50	692	710,50	710,50
CPU Média	2,06	1,72	1,54	1,3	1,25	2,88	2,634	2,62	2,23	1,85
Makespan Média	516,75	488,35	494,73	496,58	477,50	724,33	707,30	701,63	712,02	700,70

Apêndice D – Resultados dos Testes Obtidos pelo AGH

Tabela 41 – Resultados dos testes obtidos pelo AGH (Instâncias 1,2 e 3)

Instância 1		Instância 2		Instância 3	
375	0,70	415,50	0,94	461	0,94
381,67	0,81	420,50	1,05	462,50	1,14
396	0,52	421,83	1,16	461,17	1,17
378	0,69	416,50	1,20	469,17	1,41
385,67	0,81	422,50	0,95	471,50	1,19
373,50	0,58	415,17	0,88	465,50	1,28
375	0,66	420,50	1,18	477,50	1,05
376,50	1,03	416,50	1,03	467	1,19
384	0,78	419,50	0,63	473	1,41
378	0,83	415	1,09	474,50	1,11

Tabela 42- Resultados dos testes obtidos pelo AGH (Instâncias 4, 5 e 6)

Instância 4		Instância 5		Instância 6	
519,50	1,53	3217	2,59	3087,50	1,84
515,50	1,44	3097,50	2,14	3047,50	3,19
520	1,63	3072,17	1,67	3361,50	2,58
514	1,69	2982,50	2,67	3180,50	3,23
527,50	1,25	3108,50	2,52	3243,50	2,42
560	1,86	3121,50	2,25	3249	3,09
528,83	1,53	3187,50	2,11	3174,50	2,41
515,50	1,33	2983,50	2,45	3187,50	3,52
520	1,44	3197,50	2,53	3208,50	2,86
519,50	1,31	3055,50	2,20	3136,50	2,59

Tabela 43 - Resultados dos testes obtidos pelo AGH (Instâncias 7, 8 e 9)

Instância 7		Instância 8		Instância 9	
3510,50	3,06	3592,40	3,55	521	0,44
3279,90	2,34	3778,90	3,59	515	0,30
3932,30	3,50	3955,90	5,41	533	0,28
3453,30	2,39	4331,40	4,72	513,67	0,33
3219,90	2,95	3924,90	5,09	556	0,50
3232,90	2,47	4042,23	3,91	539	0,30
3392,90	2,09	3757,23	4,08	528,50	0,58
3367,90	2,92	4312,90	5,14	536	0,48
3193,90	2,30	4318,90	3,86	563	0,33
3245,30	2,34	4318,40	4,81	527	0,27

Tabela 44 - Resultados dos testes obtidos pelo AGH (Instâncias 10, 11 e 12)

Instância 10		Instância 11		Instância 12	
432	0,83	427	1,23	644	2,13
415,33	0,63	449,83	0,86	664,50	2,77
462	1,28	469,50	1	694	2,39
462	0,81	458,50	1,09	644	2,39
414,33	0,58	455,33	1,89	649,33	2,34
486,50	1,50	445	1,27	644	2,23
438	1,09	431,50	1,11	669	2,33
423,99	0,41	452,50	1,08	664,50	2,48
473	1,36	441,83	0,86	654	2,28
517	1,20	448	1,44	658	2,39

Tabela 45 - Resultados dos testes obtidos pelo AGH (Instâncias 13 e 14)

Instância 13		Instância 14	
703,33	2,06	763,50	2,88
688,33	2,92	779,33	2,98
683,33	3,34	763,50	3,42
688,33	2,56	749,33	3,13
718,33	3,05	763,50	3,61
684,33	3,19	754,33	3,14
697,17	2,77	749,33	3,52
710,50	2,64	780,67	2,58
693	2,88	794,33	2,88
703,33	2,06	763,50	2,67