OTÁVIO HENRIQUE DE SOUZA

RESOLUÇÃO DE UM PROBLEMA DE *FLOW SHOP* COM INDISPONIBILIDADE DE MÁQUINAS E MINIMIZAÇÃO DO ATRASO TOTAL

São Paulo
2025

OTÁVIO HENRIQUE DE SOUZA

RESOLUÇÃO DE UM PROBLEMA DE *FLOW SHOP* COM INDISPONIBILIDADE DE MÁQUINAS E MINIMIZAÇÃO DO ATRASO TOTAL

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

São Paulo
2025

OTÁVIO HENRIQUE DE SOUZA

RESOLUÇÃO DE UM PROBLEMA DE *FLOW SHOP* COM INDISPONIBILIDADE DE MÁQUINAS E MINIMIZAÇÃO DO ATRASO TOTAL

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

Área de Concentração:

Pesquisa Operacional

Orientadora:

Débora Pretti Ronconi

São Paulo
2025

AGRADECIMENTOS

Agradeço, em primeiro lugar, à minha professora orientadora, Dra. Débora Pretti Ronconi, pela orientação atenta e constante. Sua dedicação e disposição em esclarecer dúvidas, oferecer direcionamentos e incentivar a evolução do trabalho foram fundamentais para que este projeto se concretizasse.

Sou profundamente grato aos meus pais, Cleber e Luciana, pelo apoio incondicional ao longo de toda a minha trajetória acadêmica. Seu incentivo diário, carinho e confiança foram essenciais para que eu tivesse segurança e tranquilidade para enfrentar cada etapa da graduação.

Agradeço também aos meus amigos, que tornaram essa caminhada mais leve por meio da companhia, das conversas e do apoio nos momentos mais desafiadores.

Por fim, registro minha gratidão à Lais Dionizio, cuja contribuição ao disponibilizar as instâncias utilizadas neste estudo foi imprescindível para o desenvolvimento desta pesquisa.

"O sucesso pode vir aos humildes e aos pouco talentosos, mas a verdadeira marca de uma grande pessoa é triunfar sobre os desastres e os temores da vida humana."

— Sêneca

RESUMO

O presente trabalho investiga o problema de *flow shop* permutacional com duas máquinas, considerando tarefas retomáveis, janelas determinísticas de indisponibilidade e o critério de minimização do atraso total. Esse conjunto de características representa um cenário altamente relevante na prática industrial, mas ainda pouco explorado na literatura, dada a complexidade combinatória do problema e a dificuldade adicional introduzida pelas interrupções planejadas de máquina. Inicialmente, reproduz-se uma formulação matemática consolidada na literatura, capaz de modelar com rigor o comportamento temporal das operações sob indisponibilidades retomáveis e servir como referência para validação das soluções heurísticas.

Diante da inviabilidade de métodos exatos para instâncias de porte moderado ou grande, este trabalho desenvolve e implementa uma meta-heurística baseada em *Variable Neighborhood Search* (VNS), adaptada às particularidades do problema. O método incorpora vizinhanças clássicas, como *Swap* e *Chain Exchange*, além de um operador especializado denominado *Downtime-Aware Relocate*, criado para explorar estruturas decorrentes das janelas de indisponibilidade. A intensificação é conduzida exclusivamente pelo movimento *Relocate*, cuja eficácia para critérios de atraso é confirmada tanto pela literatura quanto pelos experimentos realizados.

Os resultados computacionais demonstram que o VNS proposto é capaz de produzir soluções de alta qualidade em diferentes portes de instância. Para as instâncias pequenas, cujas soluções ótimas são conhecidas, os *gaps* obtidos ficaram entre 4% e 7%. Para as instâncias intermediárias e grandes, comparadas ao melhor *upper bound* fornecido pelo modelo exato, os *gaps* variaram de aproximadamente 16% para 5 minutos de execução e cerca de 12% para 15 minutos. Além disso, o algoritmo exige a calibração de apenas um parâmetro, o tempo de execução, o que reduz significativamente o custo de configuração e torna a metodologia compatível com aplicações reais que demandam rapidez e previsibilidade.

Os resultados obtidos evidenciam que o VNS adaptado oferece uma combinação consistente entre eficiência computacional, simplicidade de implementação e qualidade das soluções, o que confirma sua viabilidade como ferramenta para problemas de *flow shop* com indisponibilidade de máquinas e minimização do atraso total.

Palavras-Chave – *Flow Shop*, Disponibilidade, Meta-heurística, VNS, *Tardiness*.

ABSTRACT

This work investigates the two-machine permutation *flow shop* scheduling problem with resumable tasks, deterministic machine downtime windows, and the objective of minimizing total tardiness. This combination of characteristics reflects a highly relevant industrial scenario that remains underexplored in the literature due to the combinatorial complexity of the problem and the additional challenges introduced by planned machine unavailability. A mathematical formulation commonly adopted in the literature is reproduced to rigorously model the temporal behavior of operations under resumable interruptions and to serve as a reference for validating heuristic solutions.

Given the intractability of exact methods for medium and large instances, this work develops and implements a metaheuristic based on *Variable Neighborhood Search* (VNS), adapted to the particularities of the problem. The method incorporates classical neighborhoods such as *Swap* and *Chain Exchange*, as well as a specialized operator named *Downtime-Aware Relocate*, designed to exploit structural patterns created by downtime windows. Intensification is guided exclusively by the *Relocate* movement, whose effectiveness for tardiness-based criteria is supported by the literature and by computational experiments.

Computational results show that the proposed VNS provides high-quality solutions across different instance sizes. For small instances with known optimal solutions, the obtained gaps range from 4% to 7%. For medium and large instances, when compared with the best upper bound obtained from the exact model, the gaps range from approximately 16% with 5 minutes of runtime to around 12% with 15 minutes. The algorithm also requires tuning of only one parameter, the execution time, which substantially reduces configuration effort and makes the method suitable for real-world applications that demand speed and predictability.

The results indicate that the adapted VNS offers a consistent balance between computational efficiency, implementation simplicity, and solution quality, supporting its feasibility as a tool for *flow shop* problems with machine downtime and total tardiness minimization.

Keywords – Flow Shop, Availability Constraints, Metaheuristics, VNS, Tardiness.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

A programação de tarefas é um dos temas centrais da Pesquisa Operacional e da Engenharia de Produção, presente em praticamente todos os setores industriais e de serviços. De modo geral, refere-se ao processo de definir a ordem e o momento em que diferentes trabalhos devem ser processados em um conjunto de máquinas, levando em conta limitações de capacidade, disponibilidade de recursos e critérios de desempenho. Essa problemática aparece em contextos variados, incluindo manufatura discreta, semicondutores, logística, operações hospitalares, centros de processamento de dados e sistemas de transporte (Pinedo, 2016). Ao longo das últimas décadas, a importância do tema cresceu substancialmente em função do aumento da complexidade dos sistemas produtivos, da necessidade de respostas mais rápidas às flutuações da demanda e da maior exigência dos clientes por confiabilidade e regularidade no cumprimento dos prazos.

A literatura organiza os problemas de sequenciamento a partir da combinação de três elementos fundamentais: o ambiente de máquinas, as características das tarefas e o critério de desempenho considerado. A forma como esses elementos se articulam determina tanto a complexidade do problema quanto a natureza das técnicas necessárias para resolvê-lo. Pequenas mudanças estruturais, como a introdução de datas de liberação, a existência de janelas de indisponibilidade ou a alteração do critério de otimização, podem modificar substancialmente a dificuldade do problema. Esse fenômeno é ilustrado no estudo clássico de Lenstra, Rinnooy Kan e Brucker (1977), que demonstra que muitos problemas práticos de sequenciamento pertencem à classe *NP-hard in strong sense*. Essa classificação indica que o esforço necessário para resolver o problema cresce de forma exponencial com o número de tarefas, mesmo quando os valores numéricos envolvidos são mantidos pequenos. Na prática, isso significa que não se conhecem algoritmos exatos capazes de garantir a solução ótima em tempo polinomial, e que métodos exatos se tornam rapidamente inviáveis à medida que o tamanho da instância aumenta.

Além disso, o sequenciamento envolve aspectos que vão além da escolha da ordem das tarefas. Questões relacionadas à sincronização entre máquinas, à ocorrência de ociosidade forçada, à variação nos tempos de processamento e à coordenação entre múltiplas etapas tornam o problema ainda mais complexo. Em sistemas industriais modernos, essas dinâmicas são influenciadas por fatores externos, como disponibilidade de insumos, políticas de manutenção, variações operacionais e restrições logísticas. Assim, mesmo modelos relativamente simples do ponto de vista estrutural podem apresentar comportamentos imprevisíveis quando submetidos a perturbações realistas, o que reforça a relevância de abordagens que considerem explicitamente tais restrições.

Dentro desse conjunto de modelos, o *flow shop* se destaca por representar diversos processos industriais em que os trabalhos percorrem as máquinas sempre na mesma ordem. Ele aparece em linhas de montagem sequenciais, processos de pintura e acabamento, operações de embalagem e em etapas produtivas que exigem fluxo padronizado. Uma variante particularmente relevante é o *flow shop* permutacional, no qual a mesma sequência de trabalhos deve ser respeitada em todas as máquinas. Essa característica reduz a flexibilidade de reordenação do sistema, mas mantém a complexidade combinatória alta, especialmente quando o número de tarefas cresce. Em ambientes reais, essa estrutura sequencial rígida é frequentemente observada em sistemas altamente automatizados ou em linhas cujo balanceamento depende da manutenção de uma ordem fixa entre operações.

Tradicionalmente, estudos sobre *flow shop* concentraram-se na minimização do makespan, que mede o tempo total necessário para finalizar todos os trabalhos. No entanto, o aumento da competitividade e a necessidade de atender prazos de entrega mais rigorosos tornaram os critérios associados à pontualidade cada vez mais importantes. Em sistemas sob encomenda, conhecidos como *make-to-order*, em que a produção é iniciada apenas após a realização do pedido pelo cliente, atrasos podem gerar efeitos negativos significativos. Entre esses efeitos estão penalidades contratuais, custos adicionais de reprogramação, acúmulo de estoque em processo, perda de clientes estratégicos e impactos negativos sobre a reputação da empresa. Nesse contexto, a minimização do atraso total ganha relevância por representar de maneira direta a qualidade do serviço prestado e a confiabilidade do sistema produtivo.

Outro elemento fundamental em ambientes reais é a disponibilidade das máquinas. A hipótese de máquinas continuamente disponíveis, frequente em modelos clássicos, raramente se verifica na prática. Paradas para manutenção preventiva, inspeções de rotina, ajustes operacionais, setups prolongados e falhas inesperadas criam períodos em que determinados recursos não podem processar tarefas. Essas janelas de indisponibilidade

modificam a dinâmica do sequenciamento, afetam a viabilidade de diferentes sequências e podem alterar significativamente os tempos de conclusão das operações. Em situações em que o processamento pode ser interrompido e retomado a partir do ponto em que parou, caracterizando tarefas retomáveis, o modelo torna-se ainda mais próximo da realidade industrial encontrada em setores como impressão, usinagem parcial e processos térmicos.

Por fim, a combinação entre *flow shop*, critérios de atraso e indisponibilidades cria um cenário particularmente desafiador. A presença de paradas programadas ou inesperadas pode obrigar o deslocamento de operações, alterar pontos de gargalo e modificar completamente o ordenamento ideal dos trabalhos. Apesar da relevância prática dessa interação, o número de estudos que abordam simultaneamente esses três elementos ainda é limitado, especialmente no contexto específico do *flow shop* permutacional com minimização do atraso total. Essa lacuna na literatura reforça a importância de estudos que explorem métodos eficazes para lidar com esses cenários, contribuindo tanto para o entendimento teórico do problema quanto para soluções aplicáveis em ambientes industriais reais.

1.2 Objetivos

O presente trabalho tem como objetivo central investigar a aplicação da meta-heurística Variable Neighborhood Search (VNS) ao problema de *flow shop* permutacional composto por duas máquinas, considerando tarefas retomáveis, presença de janelas de indisponibilidade e critério de minimização do atraso total. Esse problema, além de ser representativo de diversos sistemas produtivos reais, apresenta características estruturais que ampliam sua complexidade, especialmente pela interação entre a ordem fixa de processamento, as interrupções impostas pelas indisponibilidades e a avaliação do desempenho com base no atraso total acumulado. A literatura recente evidencia esse desafio em estudos como Berrais et al. (2014) e Rakrouki et al. (2023), que apresentam recortes semelhantes ao aqui analisado, ainda que com enfoques distintos em termos das soluções propostas.

Embora formulações matemáticas já tenham sido desenvolvidas para esse tipo de problema, como a proposta por Dionizio (2024), a literatura reconhece que métodos exatos tendem a enfrentar limitações severas de desempenho conforme cresce o número de tarefas ou a complexidade da estrutura de indisponibilidade. Dessa forma, técnicas heurísticas e meta-heurísticas tornam-se alternativas naturais e necessárias para problemas dessa escala. Entre essas abordagens, destaca-se o método Variable Neighborhood Search (VNS), inicialmente proposto por Mladenović e Hansen (1997), cuja estratégia de exploração sistemática de múltiplas vizinhanças e capacidade de escapar de ótimos locais reforça seu

potencial para lidar com problemas combinatórios complexos como o aqui estudado.

Diante desse cenário, o objetivo geral deste trabalho consiste em adaptar, implementar e avaliar uma versão do VNS voltada especificamente para o *flow shop* permutacional com tarefas retomáveis e janelas de indisponibilidade, investigando sua capacidade de produzir soluções de boa qualidade e seu comportamento sob diferentes configurações do problema. Para dar suporte a esse objetivo geral, o trabalho estabelece um conjunto de objetivos específicos que orientam sua condução ao longo dos capítulos.

Primeiramente, busca-se reproduzir a formulação matemática apresentada por Dionizio (2024), que descreve de forma precisa a dinâmica das tarefas em ambientes sujeitos a indisponibilidades, incorporando aspectos como retomada de operações, sincronização entre máquinas e impacto temporal das janelas de parada. Essa formulação será utilizada como referência conceitual e como base comparativa para validar as soluções produzidas pela meta-heurística em instâncias de pequeno porte, permitindo avaliar a proximidade entre os resultados heurísticos e os valores ótimos conhecidos.

Em seguida, pretende-se desenvolver uma implementação completa e funcional da meta-heurística VNS, envolvendo a definição de vizinhanças adequadas ao contexto do problema, a elaboração de operadores de perturbação que capturem explicitamente os efeitos das janelas de indisponibilidade e a criação de uma rotina de busca local fundamentada no movimento de realocação de tarefas. A integração dessas etapas deve equilibrar intensificação e diversificação ao longo da busca, assegurando a exploração de regiões promissoras do espaço de soluções sem perda da capacidade de escapar de configurações desfavoráveis.

Outro objetivo importante consiste em realizar uma análise computacional ampla e estruturada, utilizando tanto instâncias de pequeno porte, nas quais há solução ótima conhecida, quanto instâncias de maior porte, para as quais métodos exatos enfrentam limitações práticas. Essa análise permitirá avaliar a qualidade das soluções produzidas pelo VNS, seu tempo de execução, sua estabilidade diante de diferentes padrões de indisponibilidade e sua sensibilidade ao tamanho das instâncias. A comparação entre os resultados heurísticos e os valores ótimos obtidos fornecerá uma visão abrangente da eficiência da meta-heurística.

Por fim, o trabalho busca oferecer uma visão integrada de como cada etapa contribui para o entendimento do problema e para a avaliação da abordagem meta-heurística adotada. Nesse sentido, o texto está organizado da seguinte forma: o Capítulo 2 apresenta a descrição detalhada do problema estudado, formalizando suas características, restrições

e particularidades operacionais. O Capítulo 3 traz a revisão da literatura, contextualizando os principais conceitos já desenvolvidos no campo do sequenciamento, com ênfase nos estudos que abordam indisponibilidade de máquinas, tarefas retomáveis e critérios de atraso. O Capítulo 4 descreve a formulação matemática utilizada como referência, reproduzindo o modelo proposto por Dionizio (2024) e discutindo sua estrutura. O Capítulo 5 apresenta a meta-heurística VNS, detalhando as vizinhanças adotadas, os operadores de perturbação e o procedimento de busca local implementado. O Capítulo 6 apresenta os experimentos computacionais, descrevendo o conjunto de instâncias utilizadas, a metodologia de teste, os parâmetros adotados e a análise dos resultados obtidos. Finalmente, o Capítulo 7 sintetiza as conclusões do estudo, discute suas contribuições e limitações e propõe direções para pesquisas futuras.

2 DESCRIÇÃO DO PROBLEMA

O problema estudado neste trabalho consiste em um ambiente de *flow shop* composto por duas máquinas em série e um conjunto de trabalhos independentes que devem ser processados seguindo sempre a mesma ordem. Esse tipo de sistema produtivo é encontrado em diversos ambientes industriais que operam com etapas sequenciais rígidas, como linhas de pintura, processos térmicos, operações de acabamento, impressão e diferentes linhas de montagem. O problema adotado aqui representa um recorte específico dentro da ampla classe dos problemas de *flow shop* e sua caracterização requer uma compreensão das principais variantes dessa família de modelos.

Problemas de *flow shop* podem assumir três configurações clássicas. No *flow shop* não permutacional, a ordem dos trabalhos pode variar entre as máquinas, o que aumenta a flexibilidade, mas torna a modelagem mais complexa. No *flow shop* híbrido, as máquinas são organizadas em estágios que podem conter máquinas paralelas, permitindo balanceamento de carga e representando ambientes com múltiplas linhas produtivas por etapa. No flow shop permutacional, adotado neste trabalho, todos os trabalhos seguem exatamente a mesma sequência em todas as máquinas. Embora essa configuração reduza a flexibilidade operacional, ela se alinha a sistemas rigidamente encadeados e é amplamente utilizada na literatura devido à sua relevância prática e elevada complexidade combinatória.

Outro aspecto fundamental refere-se ao comportamento das tarefas diante de interrupções. A literatura distingue três categorias principais: tarefas não retomáveis, tarefas semi-retomáveis e tarefas retomáveis. Nas tarefas não retomáveis, típicas de determinados processos químicos e térmicos, qualquer interrupção invalida completamente o processamento, obrigando o reinício da operação. Nas tarefas semi-retomáveis, apenas parte do progresso é perdida e uma fração do processamento deve ser refeita. Nas tarefas retomáveis, adotadas neste estudo, a operação pode ser interrompida e retomada exatamente do ponto em que foi pausada, preservando integralmente o processamento já realizado. Essa característica é compatível com operações como usinagem parcial, fresagem, gravação, impressão e outros processos discretos.

O problema considerado também incorpora janelas de indisponibilidade determinísticas nas máquinas. Diferentemente de modelos estocásticos baseados em falhas aleatórias, aqui as indisponibilidades ocorrem em intervalos previamente conhecidos, decorrentes de manutenções planejadas, inspeções ou reconfigurações operacionais. Durante esses intervalos, nenhuma operação pode ser processada e tarefas em execução são interrompidas e retomadas após o término da indisponibilidade, coerentemente com o caráter retomável assumido. A presença dessas janelas altera significativamente a dinâmica temporal do cronograma, afetando o posicionamento das tarefas, os tempos de conclusão, a formação de gargalos e a própria sequência que minimiza o critério de desempenho.

Seguindo a estrutura geral da notação $\alpha \mid \beta \mid \gamma$ tradicionalmente utilizada na literatura de sequenciamento, o problema pode ser descrito como um caso de *flow shop* de duas máquinas cujo bloco β incorpora indisponibilidades determinísticas e tarefas retomáveis. Estudos como os de Rakrouki et al. (2023) e Berrais et al. (2014) empregam notações estendidas como $F2, hlo \mid r\text{-}a, r_j \mid \sum T_j$, nas quais o termo *hlo* indica a presença de intervalos de indisponibilidade (holes) nas máquinas e o termo $r\text{-}a$ registra que o processamento é retomável. Essa notação não faz parte da taxonomia original de Pinedo, mas representa um refinamento adotado em trabalhos que tratam especificamente de indisponibilidades determinísticas. No presente estudo, adota-se um recorte ainda mais específico, pois não são consideradas datas de liberação e o foco é exclusivamente a minimização do atraso total.

Cada trabalho J_j é caracterizado pelos tempos de processamento p_{1j} e p_{2j} nas máquinas M_1 e M_2, respectivamente, e por uma data de entrega d_j. O critério de desempenho adotado é a minimização do atraso total, definido por

$$T_j = \max\{0, C_j - d_j\},$$

onde C_j é o instante de conclusão do trabalho J_j. Se $C_j \leq d_j$, o trabalho é concluído dentro do prazo e não contribui para o atraso total. Caso contrário, o atraso corresponde ao tempo em que a data de entrega é ultrapassada. Esse critério contrasta com o *makespan*, definido por

$$C_{\max} = \max_j C_j,$$

que representa o instante de conclusão do último trabalho do cronograma. Enquanto o *makespan* mede a eficiência global do sistema, o atraso total captura de maneira direta

a pontualidade das entregas individuais. Em ambientes de produção sob encomenda, essa distinção é fundamental, pois um cronograma com *makespan* reduzido pode ainda ser inadequado se diversos trabalhos forem concluídos após suas datas de entrega.

Para ilustrar essas características, considere a instância hipotética apresentada na Tabela 1. Assume-se que a máquina M_1 apresenta uma indisponibilidade no intervalo $[2, 4]$, enquanto a máquina M_2 apresenta uma indisponibilidade no intervalo $[6, 7]$. Ambas as máquinas processam tarefas retomáveis, de modo que interrupções não implicam perda de progresso.

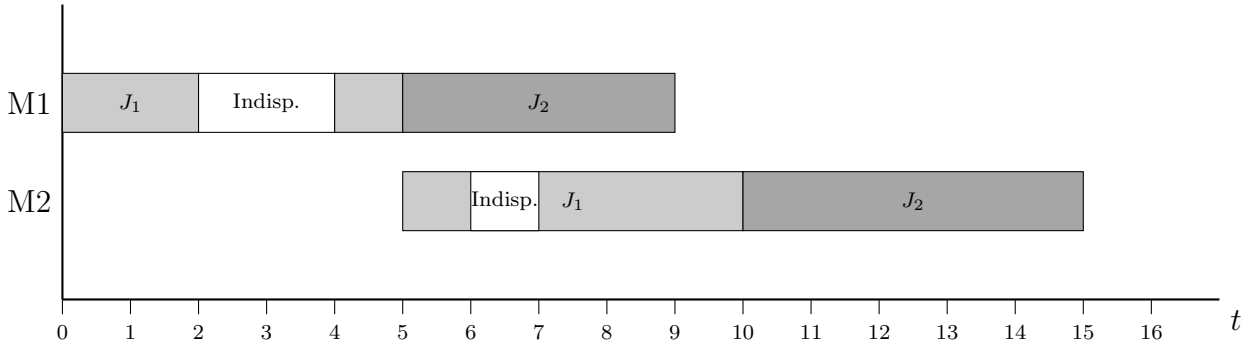
Tabela 1: Instância hipotética com $m = 2$ e $n = 2$.

Trabalho	p_{1j}	p_{2j}	d_j
J_1	3	4	10
J_2	4	5	12

Fonte: autoria própria.

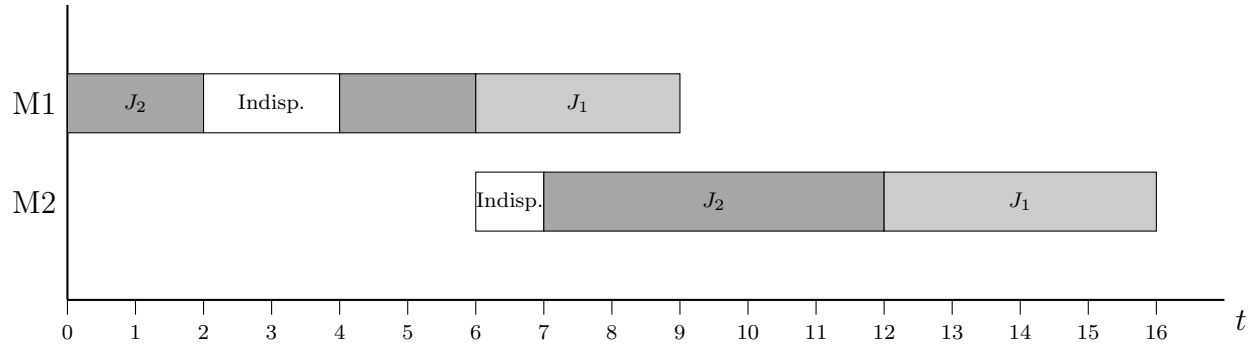
A seguir são apresentados os diagramas de Gantt correspondentes às sequências $J_1 \rightarrow J_2$ e $J_2 \rightarrow J_1$. Os gráficos destacam explicitamente os períodos de indisponibilidade e o comportamento retomável das tarefas.

Figura 1: Gráfico de Gantt para a sequência $J_1 \rightarrow J_2$.



Fonte: autoria própria.

Figura 2: Gráfico de Gantt para a sequência $J_2 \rightarrow J_1$.



Fonte: autoria própria.

A partir dos diagramas, é possível calcular os tempos de conclusão e os atrasos em cada sequência. Na sequência $J_1 \rightarrow J_2$, o trabalho J_1 é processado em M_1 no intervalo $[0, 2]$, é interrompido pela indisponibilidade de M_1 em $[2, 4]$ e retomado em $[4, 5]$, concluindo seu processamento nessa máquina em $t = 5$. Em seguida, J_2 é processado em M_1 de $t = 5$ a $t = 9$. Em M_2, J_1 inicia em $t = 5$, é afetado pela indisponibilidade de M_2 no intervalo $[6, 7]$ e termina em $t = 10$, enquanto J_2 é processado de $t = 10$ a $t = 15$. Assim, obtêm-se $C_1 = 10$ e $C_2 = 15$, o que resulta em $T_1 = \max\{0, 10-10\} = 0$ e $T_2 = \max\{0, 15-12\} = 3$, com atraso total igual a 3 unidades.

Na sequência inversa $J_2 \rightarrow J_1$, o trabalho J_2 é processado em M_1 no intervalo $[0, 2]$, sofre interrupção em $[2, 4]$ e tem seu processamento retomado em $[4, 6]$, concluindo em $t = 6$. O trabalho J_1 é então processado em M_1 de $t = 6$ a $t = 9$. Em M_2, devido à indisponibilidade no intervalo $[6, 7]$, J_2 só pode iniciar em $t = 7$, sendo processado até $t = 12$. Em seguida, J_1 ocupa M_2 de $t = 12$ a $t = 16$. Dessa forma, obtêm-se $C_2 = 12$ e $C_1 = 16$, o que leva a $T_2 = \max\{0, 12 - 12\} = 0$ e $T_1 = \max\{0, 16 - 10\} = 6$, resultando em atraso total igual a 6 unidades.

A comparação entre as duas sequências mostra que a interação entre a ordem de processamento, as janelas de indisponibilidade e o caráter retomável das tarefas pode alterar de forma significativa o desempenho do sistema em termos de atraso total. Entre os dois cenários analisados, a sequência $J_1 \rightarrow J_2$ produz atraso total igual a 3 unidades, enquanto a sequência $J_2 \rightarrow J_1$ resulta em atraso total igual a 6 unidades. Portanto, do ponto de vista da minimização do atraso total, a sequência $J_1 \rightarrow J_2$ é a melhor alternativa para a instância considerada, ilustrando de maneira concreta como a escolha da ordem de processamento impacta o resultado final do problema.

Caso o critério de desempenho fosse a minimização do *makespan*, isto é, do instante

em que o último trabalho do cronograma é concluído, a avaliação das sequências seria distinta daquela obtida com base no atraso total. Na sequência $J_1 \rightarrow J_2$, o *makespan* corresponde a $C_2 = 15$, enquanto na sequência $J_2 \rightarrow J_1$ o *makespan* é igual a $C_1 = 16$. Portanto, mesmo sob o critério de *makespan*, a sequência $J_1 \rightarrow J_2$ permaneceria como a alternativa superior para esta instância específica, embora por razões diferentes daquelas associadas ao atraso total. Essa observação reforça que o impacto das indisponibilidades e do comportamento retomável das tarefas afeta múltiplas métricas de desempenho e que a escolha do critério adotado pode alterar a interpretação dos resultados, ainda que, em alguns casos, as conclusões finais coincidam.

O exemplo apresentado permite compreender de forma clara como as características estruturais do problema (a ordem de processamento imposta pelo ambiente *flow shop*, as janelas de indisponibilidade nas máquinas, o comportamento retomável das tarefas e a adoção do atraso total como critério de desempenho) interagem para moldar a complexidade do sequenciamento. Mesmo em instâncias pequenas, variações aparentemente simples na sequência dos trabalhos produzem efeitos substanciais nos tempos de conclusão e no valor da função objetivo, o que evidencia o caráter combinatório e desafiador do problema. Diante desse cenário, torna-se necessário recorrer a modelos formais que descrevam o comportamento temporal do sistema de maneira rigorosa, bem como ao desenvolvimento de métodos de solução capazes de explorar de forma eficiente o espaço de sequências viáveis. As próximas seções apresentam a formulação matemática utilizada como referência para validação e, em seguida, a meta-heurística baseada em vizinhança variável desenvolvida neste trabalho.

3 REVISÃO BIBLIOGRÁFICA

A presente revisão bibliográfica está organizada de maneira a conduzir o leitor do panorama mais abrangente da área de programação de atividades *(scheduling)* até o problema específico abordado neste trabalho. Inicialmente, apresentam-se conceitos fundamentais e resultados clássicos relacionados à complexidade dos problemas de programação de tarefas, com destaque para a distinção entre formulações polinomialmente solúveis e aquelas classificadas como *NP-hard*. Em seguida, examina-se o modelo de *flow shop*, um dos mais investigados na literatura de programação da produção, detalhando suas variantes e critérios de desempenho mais comuns. A discussão é então direcionada para trabalhos que tratam da minimização do atraso total (tardiness), e posteriormente para pesquisas que incorporam restrições de disponibilidade das máquinas. Por fim, o texto concentra-se nas contribuições que se aproximam diretamente do problema em estudo, evidenciando a escassez de pesquisas nessa interseção e reforçando a relevância da investigação proposta.

3.1 Problemas de *Scheduling*

Os problemas de scheduling constituem uma das áreas centrais da Pesquisa Operacional, dada sua presença em diferentes setores industriais e de serviços e seu impacto direto sobre custos operacionais, utilização de recursos e atendimento a prazos. Esses problemas envolvem determinar a ordem e o instante em que cada trabalho deve ser processado em um conjunto de recursos limitados, respeitando restrições do sistema e otimizando critérios que refletem eficiência, regularidade ou pontualidade. A literatura clássica organiza esses modelos por meio da notação $\alpha \mid \beta \mid \gamma$, introduzida por Graham et al. (1979) e amplamente difundida em obras como Pinedo (2016), a qual permite classificar o ambiente produtivo, as características adicionais e o critério de desempenho de maneira estruturada.

No que diz respeito ao ambiente α, diferentes famílias de modelos são estudadas devido às suas propriedades estruturais e aplicações práticas. Os problemas de uma

única máquina constituem a base para grande parte da teoria e já apresentam desafios consideráveis quando critérios associados a prazos são introduzidos. Em ambientes com máquinas paralelas, o problema envolve decisões de alocação e balanceamento de carga, o que amplia o espaço de busca. Os modelos job shop permitem que cada trabalho siga uma rota própria, resultando em uma grande flexibilidade combinatória. Os modelos open shop vão além, pois não impõem ordem fixa entre as máquinas. Os ambientes *flow shop*, por sua vez, representam sistemas em que todos os trabalhos percorrem as máquinas na mesma ordem, o que lhes confere uma estrutura intermediária entre simplicidade e complexidade e os torna adequados para descrever processos produtivos compostos por etapas sequenciais.

A dimensão γ, referente ao critério de desempenho, também exerce papel significativo na formulação e na complexidade do modelo. Critérios como o *makespan*, que mede o tempo total necessário para concluir todos os trabalhos, e o fluxo médio, que avalia a rapidez com que as operações são finalizadas, são amplamente estudados na literatura. Em ambientes com prazos rígidos, critérios associados à pontualidade, como atraso total, número de trabalhos atrasados ou medidas de earliness–tardiness, tornam-se particularmente relevantes. A escolha do critério pode modificar substancialmente o comportamento do problema, dado que muitos modelos resolvíveis sob *makespan* tornam-se *NP-hard* quando o critério passa a envolver prazo, como é o caso da minimização do atraso total.

A dimensão β, que incorpora características adicionais e restrições operacionais, engloba uma ampla variedade de elementos encontrados na prática industrial. Entre as restrições mais investigadas estão tempos de setup dependentes da sequência, ausência de buffers intermediários, restrições do tipo no-wait, datas de liberação, janelas de tempo e elegibilidade restrita de máquinas. A presença ou ausência de preempção e retomabilidade afeta diretamente o comportamento do modelo, assim como restrições relacionadas à disponibilidade das máquinas, como janelas de indisponibilidade determinísticas ou estocásticas. Essas características ampliam o espaço de busca e influenciam a viabilidade e a qualidade das sequências de processamento.

A compreensão formal da dificuldade desses modelos foi estabelecida pelo estudo seminal de Lenstra, Rinnooy Kan e Brucker (1977), que demonstrou que pequenas modificações no ambiente ou no critério podem transformar problemas polinomiais em problemas NP-hard. O conceito de *NP-hard* in strong sense, introduzido pelos autores, é fundamental para entender a complexidade intrínseca de diversos modelos de scheduling. Um problema classificado dessa forma permanece *NP-hard* mesmo quando todos os parâmetros

numéricos são pequenos ou representados em formato unário, o que indica que a dificuldade decorre de sua estrutura combinatória e não apenas da magnitude dos números envolvidos. Como consequência, algoritmos pseudo-polinomiais não são capazes de resolver o problema de forma eficiente e métodos exatos tornam-se inviáveis para instâncias de porte moderado. Essa constatação fundamenta o uso de heurísticas e meta-heurísticas, decisivas para a obtenção de soluções de boa qualidade dentro de tempos computacionais aceitáveis.

O ambiente *flow shop* é um dos mais estudados dentro dessa literatura. Na formulação clássica com duas máquinas e critério *makespan*, a Regra de Johnson, proposta em 1954, fornece uma sequência ótima por meio de uma construção simples baseada nos tempos de processamento de cada trabalho. Trabalhos cujo tempo na primeira máquina é menor ou igual ao da segunda são posicionados no início da sequência em ordem crescente de p_{1j}. Trabalhos cujo tempo na primeira máquina é maior que o da segunda são posicionados ao final da sequência em ordem decrescente de p_{2j}. Essa estrutura permite resolver o problema de maneira eficiente e ilustra como, sob condições específicas, a busca pela solução ótima pode ser conduzida por regras determinísticas. Entretanto, a introdução de uma terceira máquina elimina essa propriedade e transforma o problema em NP-hard. O mesmo ocorre quando se adota critérios alternativos, como atraso total, ou quando se incorporam restrições mais realistas, como indisponibilidades ou tempos de setup dependentes da sequência.

Esse panorama evidencia que problemas de scheduling, mesmo quando aparentemente simples, podem apresentar elevada complexidade e exigir técnicas avançadas de busca. A combinação entre diferentes ambientes, critérios e restrições produz modelos cujo comportamento é difícil de analisar e cujo espaço de soluções cresce rapidamente com o número de trabalhos. Esse contexto motiva o uso de meta-heurísticas como ferramenta essencial para explorar de forma eficiente o espaço de busca, especialmente em problemas que integram características como *flow shop* permutacional, tarefas retomáveis, janelas de indisponibilidade e critério de atraso total, que constituem o foco deste trabalho.

3.2 Resolução de Problemas de *Flow Shop*

O ambiente de *flow shop* apresenta uma ampla variedade de estruturas e comportamentos e, por isso, ocupa posição de destaque na literatura de sequenciamento. Embora todos os trabalhos sigam a mesma ordem de passagem pelas máquinas, diferentes configurações produzem dinâmicas operacionais distintas e influenciam diretamente tanto a

formulação quanto a dificuldade do problema. Variantes como o *flow shop* puro, o *flow shop* não permutacional, o *flow shop* híbrido com máquinas paralelas por estágio e o *flow shop* com *buffers* intermediários limitados ou inexistentes são amplamente discutidas em obras clássicas como Pinedo (2016). Além disso, modelos específicos como o *flow shop no-wait*, caracterizado pela proibição de esperas entre máquinas, aparecem com frequência em processos metalúrgicos e químicos, reforçando a diversidade estrutural que esse ambiente comporta.

A escolha do critério de desempenho exerce papel central na caracterização do problema. Critérios como *makespan*, atraso máximo, atraso total e medidas combinadas de *earliness* e *tardiness* alteram de maneira significativa o comportamento do sistema. Enquanto o *makespan* privilegia a eficiência global da produção, critérios associados a prazos tornam o problema extremamente sensível à posição relativa dos trabalhos na sequência. Essa diferença explica por que o *flow shop* com duas máquinas e critério *makespan* admite solução ótima via Regra de Johnson (Johnson, 1954), enquanto a inclusão de uma terceira máquina ou a substituição do critério por atraso total destrói completamente essa estrutura ordenada. Como demonstrado por Garey, Johnson e Sethi (1976), o *flow shop* permutacional com três máquinas e critério *makespan* já é *NP-hard*, e o aumento da complexidade é ainda mais acentuado quando se adotam critérios baseados em prazos, que costumam gerar paisagens de busca altamente irregulares.

Além das variantes estruturais e dos critérios, diversas extensões do *flow shop* enriquecem o modelo e o aproximam de sistemas reais. Pinedo (2016) compila extensões como tempos de *setup* dependentes da sequência, datas de liberação, janelas de tempo, preempção, retomabilidade do processamento e indisponibilidades de máquina decorrentes de manutenções planejadas ou intervenções operacionais. Essas extensões eliminam propriedades estruturais que poderiam ser exploradas por métodos determinísticos, ampliam o espaço de busca e reforçam a necessidade de métodos aproximados. A combinação entre diferentes restrições impõe desafios adicionais e explica por que heurísticas, abordagens híbridas e meta-heurísticas se tornaram centrais no tratamento de variantes práticas do *flow shop*.

No contexto específico de critérios baseados em prazos, o estudo pioneiro de Kim (1993) representou um divisor de águas ao propor heurísticas voltadas para a minimização do atraso médio em *flow shops*. Entre as técnicas analisadas destacaram-se adaptações do algoritmo NEH (*Nawaz–Enscore–Ham*), heurística construtiva clássica do *flow shop* inicialmente concebida para o *makespan*, e procedimentos de *local search*. O NEH parte de uma ordenação inicial dos trabalhos baseada em regras de prioridade, como soma dos

tempos de processamento ou ordem crescente de datas de entrega, e insere gradualmente cada novo trabalho na posição que minimize o critério desejado. A *local search* complementa essa abordagem ao explorar soluções vizinhas por meio de operações simples, como trocas de pares de trabalhos ou inserções. Os resultados de Kim mostraram que a combinação entre heurísticas construtivas e *local search* produzia soluções claramente superiores às tradicionais regras de despacho, estabelecendo uma nova referência para algoritmos dedicados ao *tardiness* em *flow shops*.

A consolidação do uso de meta-heurísticas para o critério de atraso total ocorreu com o trabalho de Armentano e Ronconi (1999), que desenvolveram uma *tabu search* adaptada ao *flow shop* permutacional. A *tabu search* distingue-se por permitir movimentos que não necessariamente melhoram a solução corrente, enquanto utiliza estruturas de memória de curto prazo para impedir o retorno a soluções já visitadas. Os autores combinaram mecanismos de intensificação, destinados a aprofundar a busca em regiões promissoras, e mecanismos de diversificação, responsáveis por explorar áreas pouco visitadas do espaço de soluções. Essa abordagem mostrou desempenho significativamente superior ao de heurísticas tradicionais, alcançando soluções ótimas em instâncias pequenas e apresentando robustez para instâncias médias e grandes.

Avançando nessa linha de pesquisa, Hasija e Rajendran (2004) aplicaram o *simulated annealing* ao mesmo problema. Inspirado no processo físico de resfriamento de sólidos, o *SA* aceita temporariamente soluções de pior qualidade em estágios iniciais, permitindo que a busca escape de ótimos locais. À medida que a temperatura diminui, a busca torna-se mais seletiva. Os autores enriqueceram esse arcabouço com mecanismos específicos de perturbação, como *JSB* (*job-shift-based*), *PSS* (*probabilistic-step-swap*) e o esquema *JIBIS* (*job-index-based insertion scheme*), que ampliaram a capacidade do algoritmo de explorar regiões diversas do espaço de busca. Os resultados indicaram melhorias substanciais em comparação com heurísticas clássicas, reforçando a eficácia do *SA* em cenários com critério de atraso.

Em uma etapa posterior, Vallada, Ruiz e Minella (2008) realizaram um estudo comparativo abrangente envolvendo quarenta heurísticas e meta-heurísticas distintas, sob condições experimentais uniformes. O uso de um *benchmark* padronizado e de métricas estatísticas robustas permitiu uma análise rigorosa da eficácia dos métodos. Os resultados apontaram que técnicas como *simulated annealing* e *tabu search* apresentavam desempenho superior para o critério de atraso total, enquanto heurísticas simples e adaptações diretas de algoritmos originalmente projetados para o *makespan* mostraram desempenho inferior. Esse estudo consolidou a compreensão sobre quais métodos eram efetivamente

promissores para o problema.

Por fim, Vallada e Ruiz (2010) propuseram uma família de algoritmos genéticos aprimorados, incorporando componentes como *path relinking*, mecanismos de controle de diversidade e *accelerated local search*. Os algoritmos genéticos utilizam operadores inspirados na evolução natural, como seleção, *crossover* e mutação, para explorar e combinar diferentes soluções. No estudo, a introdução do *path relinking*, que explora trajetórias entre soluções de alta qualidade, o uso de reinício controlado para evitar convergência prematura e procedimentos de *local search* com aceleração permitiram obter resultados superiores aos métodos anteriores. Esses avanços estabeleceram um novo patamar de desempenho para a minimização do *tardiness* em ambientes de *flow shop*.

3.3 *Flow Shop* com Critérios de Atraso

O estudo do critério de atraso total ($\sum T_j$) em ambientes de *flow shop* ganhou destaque a partir de contribuições que demonstraram sua relevância prática em sistemas de produção sob encomenda (*make-to-order*), nos quais o cumprimento dos prazos de entrega é determinante para a satisfação dos clientes e para a competitividade das empresas. Diferentemente do critério de *makespan*, tradicionalmente adotado na literatura, a minimização do atraso total representa um desafio adicional, pois envolve a relação entre os tempos de conclusão e as datas de entrega de cada trabalho, tornando o problema ainda mais complexo do ponto de vista computacional.

Um dos primeiros avanços relevantes nesse campo foi apresentado por Sen, Dileepan e Gupta (1989), que analisaram o problema de duas máquinas. Os autores derivaram condições de precedência ótimas entre trabalhos adjacentes e desenvolveram um algoritmo *branch-and-bound* capaz de resolver instâncias de pequeno porte. Embora restrito em termos de aplicabilidade, esse trabalho estabeleceu a base para os estudos posteriores, evidenciando a dificuldade inerente do problema e a necessidade de técnicas mais sofisticadas.

Na mesma linha de pesquisa exata, Schaller (2005) propôs melhorias significativas ao algoritmo de *branch-and-bound* para o mesmo problema. O autor revisou condições de dominância existentes, apresentou versões mais fortes e introduziu uma nova condição adicional, além de propor um limite inferior mais rigoroso para estimar o atraso em sequências parciais. Essas contribuições resultaram em algoritmos mais eficientes, capazes de resolver instâncias de maior porte em tempos computacionais reduzidos, ampliando o

alcance dos métodos exatos para esse critério.

No campo das heurísticas, Koulamas (1997) desenvolveu o chamado algoritmo *shifting bottleneck*, adaptado ao problema de *flow shop* com atraso total. A proposta buscou estruturar regras heurísticas capazes de lidar com a complexidade do problema de forma prática, ainda que sem garantia de otimalidade. Posteriormente, Koulamas (2020) aprofundou essa linha de investigação ao estudar o problema em ambientes proporcionais (*proportionate flow shop*), nos quais todos os trabalhos apresentam tempos de processamento iguais em cada máquina. O autor demonstrou como adaptar o algoritmo de programação dinâmica de Lawler (1977), originalmente concebido para máquina única, ao contexto de múltiplas máquinas proporcionais, identificou novos casos solucionáveis em tempo polinomial e propôs um esquema de aproximação totalmente polinomial (FPTAS). Esse trabalho ampliou a compreensão teórica sobre o problema e corrigiu formulações anteriores, reforçando a fronteira entre casos tratáveis e intratáveis.

A partir da década de 2010, a pesquisa concentrou-se no desenvolvimento de meta-heurísticas robustas capazes de lidar com instâncias de grande porte. Karabulut (2016) propôs um algoritmo híbrido do tipo *iterated greedy* (IG_RLS), que combina as fases de destruição e reconstrução da meta-heurística clássica com um procedimento de busca local aleatória e um mecanismo de aceleração de cálculos parciais. Além disso, introduziu uma nova fórmula de temperatura no critério de aceitação, adaptada especificamente ao objetivo de atraso total. O algoritmo estabeleceu novas melhores soluções conhecidas para centenas de instâncias, consolidando-se como um marco na evolução metodológica do problema.

Na sequência, Fernandez, Valente e Framinan (2018) realizaram uma das comparações mais abrangentes da literatura, reimplementando algoritmos de ponta sob condições uniformes e avaliando seu desempenho em larga escala. Os autores propuseram heurísticas construtivas baseadas em *beam search* e oito variantes de algoritmos *iterated greedy*, destacando-se o método IARAS, que utiliza perturbações simples por trocas adjacentes. Os resultados mostraram que tanto o *beam search* quanto o IARAS superaram consistentemente os métodos até então considerados estado da arte, incluindo algoritmos genéticos e versões anteriores de *iterated greedy*, estabelecendo um novo patamar de desempenho para o critério de atraso total em *flow shops*.

Mais recentemente, Oujana et al. (2021) ampliaram a investigação para ambientes industriais realistas, propondo um modelo de programação linear inteira mista (MILP) para o problema de *hybrid flexible flow shop* (HFFS) com o objetivo de minimizar o atraso to-

tal. O modelo incorpora restrições adicionais raramente tratadas simultaneamente, como tempos de *setup* dependentes da sequência, elegibilidade de máquinas, precedência entre trabalhos e divisão de tarefas (*job splitting*). Os resultados computacionais mostraram que o modelo é altamente eficaz para instâncias pequenas e médias, mas torna-se inviável para instâncias maiores, confirmando a limitação dos métodos exatos em problemas de grande escala e reforçando a necessidade do desenvolvimento de heurísticas e meta-heurísticas para esses cenários.

Em conjunto, essas contribuições demonstram a evolução do estudo do atraso total em *flow shops*, desde as formulações exatas iniciais até o desenvolvimento de heurísticas estruturais, meta-heurísticas avançadas e modelos aplicados a ambientes industriais complexos. Esse percurso evidencia tanto a relevância prática do critério de *tardiness* quanto os desafios computacionais que justificam a busca contínua por métodos aproximados mais eficazes.

3.4 *Flow Shop* com Restrições de Disponibilidade

Grande parte da literatura sobre problemas de *flow shop* assume implicitamente que as máquinas estão sempre disponíveis para processamento. No entanto, em ambientes produtivos reais, essa hipótese raramente se sustenta. Máquinas frequentemente precisam passar por manutenções preventivas, reparos, *setups* ou ajustes programados, gerando intervalos nos quais não estão disponíveis para operação. Essas janelas de indisponibilidade, conhecidas como *availability constraints*, aumentam significativamente a complexidade do problema de sequenciamento, pois a solução precisa respeitar não apenas a ordem de processamento entre as máquinas, mas também os períodos nos quais cada recurso pode ou não estar ativo.

Os primeiros trabalhos que sistematizaram o estudo desse tipo de restrição foram revisões conceituais. Schmidt (2000) apresentou uma taxonomia detalhada dos problemas de agendamento com disponibilidade limitada, distinguindo ambientes de máquina única, máquinas paralelas e *flow shops*, e introduzindo a notação $\alpha|\beta|\gamma$ para caracterizar máquinas, trabalhos e critérios de desempenho. Entre os conceitos centrais, destacou-se a diferenciação entre cenários *resumable*, nos quais operações interrompidas podem ser retomadas, e *non-resumable*, em que trabalhos interrompidos devem ser reiniciados do zero. Posteriormente, Ma et al. (2010) ampliaram essa revisão, abrangendo também ambientes *open shop* e *job shop*. Além de consolidar resultados de complexidade e algoritmos de aproximação, o estudo reforçou que muitos problemas simples tornam-se *NP-hard* com a

introdução de um único intervalo de indisponibilidade, em especial no caso não-resumível, e identificou lacunas de pesquisa, como a escassez de estudos sobre problemas *online* ou sobre o caso *semi-resumable*.

Na linha de propostas heurísticas, Lee (1997) foi pioneiro ao estudar o problema de duas máquinas com indisponibilidade conhecida, provando seu catáter *NP-hard* e propondo heurísticas baseadas em Johnson adaptadas para os casos em que a restrição ocorre em M1 ou M2. O autor demonstrou que a simples aplicação da regra de Johnson fornece limites de erro pouco satisfatórios, e desenvolveu heurísticas aprimoradas (H2 e H4) com garantias formais de desempenho: no caso de restrição em M1, *bound* de 1/2; e no caso de restrição em M2, *bound* de 1/3, ambos *tight*. Cheng e Wang (2000) avançaram nessa direção ao analisar a heurística de Lee e comprovar que seu *bound* era justo, além de propor uma heurística melhorada (HI) capaz de reduzir o limite de erro no pior caso para 1/3, garantindo *makespan* de até 4/3 vezes o ótimo com tempo computacional $O(n \log n)$. Breit (2004), por sua vez, investigou o caso em que a indisponibilidade ocorre na máquina B, propondo o Algoritmo H, uma heurística de aproximação eficiente que combina múltiplas sequências de referência e alcança um *bound* ainda mais restrito, de 5/4, superando o resultado de Lee.

Em paralelo, alguns trabalhos buscaram métodos exatos ou aproximações sofisticadas. Allaoui et al. (2006) analisaram o *flow shop* de duas máquinas com indisponibilidade programada na primeira máquina e estudaram a aplicação da regra de Johnson em cenários resumíveis e não-resumíveis. Embora tenham sugerido condições de otimalidade e um modelo de programação dinâmica para resolver o problema, uma errata publicada posteriormente por Rapine (2013) mostrou que as condições propostas não eram válidas e que a análise de complexidade estava incorreta, revelando limitações importantes do modelo. Wang e Cheng (2007) ampliaram a fronteira de pesquisa ao considerar tempos de *setup* separados e antecipatórios em um *flow shop* de duas máquinas com indisponibilidade em M1. Para esse problema altamente complexo, desenvolveram um esquema de aproximação em tempo polinomial (PTAS), garantindo soluções arbitrariamente próximas do ótimo. Esse resultado representou um avanço teórico expressivo na literatura de agendamento com restrições de disponibilidade.

Dada a dificuldade de métodos exatos, parte da literatura explorou o uso de metaheurísticas e aplicações específicas. Blazewicz et al. (2001) estudaram o problema de duas máquinas com períodos de indisponibilidade pré-determinados, propondo uma combinação entre heurísticas construtivas (como a regra de Johnson e uma heurística *look-ahead*) e *simulated annealing*. Os experimentos mostraram que a combinação de métodos

construtivos com busca local é robusta, especialmente em instâncias difíceis. Ben Chihaoui et al. (2010), por sua vez, analisaram o caso em que a primeira máquina requer uma manutenção preventiva flexível e os trabalhos são não-resumíveis. Os autores propuseram uma formulação MILP, um algoritmo *branch and bound* reforçado por propriedades de dominância e limites inferiores, e uma heurística rápida baseada no algoritmo NEH. Os resultados computacionais demonstraram que o *branch and bound* é eficaz para instâncias de porte médio, enquanto a heurística fornece soluções de alta qualidade em tempo muito reduzido.

Em síntese, a literatura sobre *flow shop* com restrições de disponibilidade evoluiu de forma semelhante àquela observada em problemas com critério de atraso: inicialmente marcada por modelos exatos e heurísticas construtivas, avançou para heurísticas com garantias formais e, finalmente, incorporou meta-heurísticas e esquemas de aproximação sofisticados. Apesar dos progressos, o espaço de pesquisa ainda apresenta lacunas importantes, sobretudo quando se combina indisponibilidade de máquinas com critérios de atraso, como o *tardiness*, tema que será explorado na próxima subseção.

3.5 *Flow Shop* com *Jobs* Retomáveis e Critério de Atraso

Embora a maior parte da literatura sobre *flow shop* com restrições de disponibilidade tenha como foco o critério de *makespan*, dois trabalhos recentes exploraram de forma específica o critério de atraso total (*total tardiness*) em ambientes com máquinas sujeitas a indisponibilidades. Essa característica é fundamental para este trabalho, pois o não cumprimento de prazos de entrega está diretamente associado a custos adicionais, perda de reputação e insatisfação de clientes. Outro ponto essencial é que ambos os estudos consideram o cenário de *jobs resumable*, isto é, operações interrompidas por um período de indisponibilidade podem ser retomadas a partir do ponto em que foram paradas. Essa hipótese aproxima o problema das condições industriais reais e delimita claramente o recorte da presente pesquisa.

O primeiro estudo dedicado ao problema é o de Berrais et al. (2014), que analisou o *two-machine permutation flow shop* com indisponibilidades determinísticas, datas de liberação e *jobs* retomáveis. Os autores propuseram uma formulação de programação inteira mista (MILP), mas, devido à complexidade *NP-hard* do problema, concentraram-se em heurísticas construtivas inspiradas no algoritmo NEH (Nawaz, Enscore e Ham, 1983). O NEH clássico é um método de inserção iterativa: inicialmente os trabalhos são

ordenados por uma regra de prioridade; em seguida, os dois primeiros são agendados em ambas as ordens possíveis; a partir do terceiro trabalho, cada novo job é inserido em todas as posições possíveis da sequência parcial, escolhendo-se aquela que minimiza o critério considerado. Essa lógica foi adaptada por Berrais et al. com diferentes regras de ordenação: *Earliest Due Date* (EDD), *Modified Due Date* (MDD), *Slack Time* (ST) e *Slack Time per Remaining Work* (STRW). A quinta heurística (H5) consistiu em aplicar as quatro anteriores e selecionar a melhor solução obtida. Os resultados computacionais, testando instâncias de até 500 jobs, mostraram que a heurística H5 foi consistentemente superior, com desvios relativos médios praticamente nulos em relação às melhores soluções conhecidas, mesmo em instâncias de grande porte. Esse trabalho representou, portanto, a primeira abordagem efetiva para o problema de indisponibilidade em *flow shop* com critério de atraso total.

A pesquisa foi retomada por Rakrouki et al. (2022), que ampliaram a linha anterior ao propor meta-heurísticas populacionais híbridas para o mesmo problema. Cinco algoritmos evolucionários foram desenvolvidos: *Particle Swarm Optimization* (PSO), *Differential Evolution* (DE), *Genetic Algorithm* (GA), *Ant Colony Optimization* (ACO) e *Imperialist Competitive Algorithm* (ICA). Todos foram combinados com procedimentos de busca local baseados no NEH, de modo a equilibrar exploração global e intensificação local. No PSO, partículas representavam sequências de jobs, atualizadas por regras de velocidade e posição, sendo as soluções transformadas em permutações por esquemas de ordenação. O DE explorava diferenças entre soluções para gerar novas combinações, enquanto o GA utilizava operadores de seleção, cruzamento e mutação sobre populações de sequências. O ACO construía soluções incrementalmente, guiado por trilhas de feromônio e heurísticas de distância entre jobs, e o ICA simulava competição imperialista–colônia para explorar regiões promissoras do espaço de busca. Em todos os casos, procedimentos de inserção local baseados no NEH reforçaram a qualidade das soluções finais.

Os experimentos envolveram 3.240 instâncias, variando de 10 a 300 jobs, com diferentes fatores de atraso e dispersão de datas de entrega. Os resultados indicaram que o ICA foi o algoritmo mais eficaz, obtendo as melhores soluções em quase metade das instâncias, embora com custo computacional elevado. O DE destacou-se pelo bom equilíbrio entre tempo de execução e qualidade em instâncias pequenas e médias, enquanto o GA híbrido se mostrou competitivo em problemas de maior porte. Já PSO e ACO apresentaram desempenho intermediário, com resultados estáveis, mas inferiores aos do ICA.

A comparação entre os dois trabalhos revela a evolução natural da literatura: partindo de formulações matemáticas e heurísticas construtivas de baixo custo computacional, a

pesquisa avançou para meta-heurísticas sofisticadas capazes de lidar com instâncias de maior dimensão. Ainda assim, o campo permanece incipiente, com apenas duas contribuições diretas para o problema de *flow shop* com indisponibilidade, *jobs resumíveis* e critério de atraso total. Essa lacuna reforça a relevância da presente pesquisa, que busca aprofundar esse recorte específico e propor novos modelos e abordagens para lidar com o problema.

3.6 *Variable Neighborhood Search (VNS)*

A meta-heurística Busca em Vizinhança Variável (Variable Neighborhood Search – VNS) foi introduzida por Hansen e Mladenović (1997, 2001) como um método simples e eficaz para lidar com problemas de otimização combinatória e global. Sua ideia central é a mudança sistemática da estrutura de vizinhança ao longo do processo de busca, o que permite escapar de ótimos locais e explorar regiões mais amplas do espaço de soluções. Diferente de outras meta-heurísticas que dependem fortemente de parâmetros ou estruturas fixas, a VNS combina três pilares fundamentais: (i) ótimos locais em uma vizinhança podem não ser ótimos em outra, (ii) o ótimo global é ótimo em relação a todas as vizinhanças, e (iii) mínimos locais de qualidade tendem a estar próximos, o que favorece a exploração iterativa entre vizinhanças.

O algoritmo básico do VNS é composto por três etapas principais: (i) shaking, que gera uma solução vizinha da atual de forma aleatória; (ii) busca local, aplicada sobre essa solução para encontrar um ótimo local; e (iii) mudança de vizinhança, em que a busca retorna à primeira vizinhança se há melhoria ou avança para vizinhanças mais distantes se não há progresso. Esse esquema simples pode ser estendido de diversas formas, resultando em variantes como a VNS Reduzida (RVNS), a Descida por Vizinhança Variável (VND), a VNS Básica (BVNS) e a VNS Geral (GVNS), além de adaptações como a Skewed VNS, a VNS Paralela (PVNS) e a VNS baseada em decomposição (VNDS). Esse conjunto de variantes foi sistematizado e detalhado por Mladenović, Hansen e Pérez-Brito (2009), que também catalogaram aplicações da VNS em uma vasta gama de problemas, incluindo roteamento de veículos, problemas de grafos, localização de instalações, bioinformática e scheduling.

A consolidação do VNS como uma das principais meta-heurísticas da literatura foi reforçada pelo levantamento de Brimberg et al. (2008), que analisou de forma abrangente centenas de aplicações em diferentes domínios. O estudo destacou a simplicidade, flexibilidade e robustez da VNS, bem como sua capacidade de ser hibridizada com ou-

tras abordagens, como Algoritmos Genéticos, Simulated Annealing e Busca Tabu. Desde então, a VNS tem se mostrado uma ferramenta particularmente poderosa para problemas de agendamento, devido à sua habilidade de explorar múltiplas vizinhanças e de ser facilmente adaptada a diferentes restrições industriais.

Entre as aplicações em problemas de scheduling, destaca-se o trabalho de Mladenović, Hansen e Moreno Pérez (2008), que aplicaram a GVNS a um problema de máquinas paralelas na indústria de semicondutores, com critérios de tardiness e risco de qualidade associados ao estado de saúde dos equipamentos. O estudo mostrou que o GVNS não apenas superou métodos clássicos como a Busca Tabu, mas também se beneficiou da inclusão de vizinhanças especializadas, como uma baseada em Branch-and-Bound, para lidar com as especificidades do problema. Esse resultado reforça a versatilidade do VNS em ambientes complexos e realistas.

Outro exemplo relevante é o algoritmo Iterated Greedy (IG) proposto por Ruiz e Stützle (2007) para o problema de *permutation flow shop*. Embora não seja formalmente classificado como VNS, o IG compartilha a mesma filosofia de perturbação e intensificação, alternando fases de destruição e reconstrução de soluções com busca local baseada no NEH. O algoritmo se destacou pela simplicidade e desempenho, superando algoritmos muito mais sofisticados e consolidando-se como um dos métodos de referência para o problema de *flow shop*. A proximidade conceitual entre o IG e o VNS ilustra como os princípios de vizinhança variável permeiam diversas abordagens de scheduling.

De forma geral, a literatura mostra que o VNS e suas variantes são amplamente utilizados tanto como algoritmos independentes quanto como componentes de meta-heurísticas híbridas. A facilidade de integração em frameworks evolutivos ou construtivos faz com que a VNS seja uma ferramenta natural para o desenvolvimento de algoritmos mais sofisticados, com aplicações em problemas de *flow shop*, job shop e máquinas paralelas. Apesar de sua relevância, ainda não há registros de aplicação do VNS em problemas de flow shop com indisponibilidade, jobs resumíveis e critério de atraso total, o que reforça a lacuna explorada neste trabalho.

4 FORMULAÇÃO MATEMÁTICA

O problema tratado consiste em um *flow shop* de duas máquinas com ocorrência de períodos de indisponibilidade do tipo *resumable*. Nesse cenário, se uma operação for interrompida por indisponibilidade, ela poderá ser retomada do ponto em que parou, sem necessidade de reinício. O objetivo do modelo é minimizar o atraso total dos jobs em relação a seus prazos de entrega.

A formulação matemática utilizada neste trabalho é a mesma proposta por Dionizio (2024), aplicada ao problema de minimização do atraso total em *flow shop* com indisponibilidades e tarefas retomáveis.

4.1 Elementos da Formulação

Nesta seção, apresentam-se os elementos necessários para a formalização do modelo matemático, incluindo os índices, parâmetros e variáveis utilizados. Esses componentes servem como base para a construção das restrições e para a definição da função objetivo que descreve o comportamento do problema estudado.

4.1.1 Índices

Os índices apresentados nesta subseção definem a estrutura básica do modelo, identificando operações, jobs, máquinas e intervalos de indisponibilidade. Eles permitem organizar a formulação de forma sistemática, garantindo que cada componente do problema seja referenciado de maneira clara e consistente ao longo das equações. A seguir estão detalhados os índices do modelo.

i operações do conjunto de jobs

j jobs

k máquinas

l períodos de indisponibilidade da máquina k

4.1.2 Parâmetros

Os parâmetros representam as informações de entrada do problema, como tempos de processamento, datas de entrega e janelas de indisponibilidade das máquinas. Esses elementos descrevem o ambiente operacional do *flow shop* e fornecem os valores necessários para que o modelo capture corretamente as restrições e a dinâmica temporal do sistema. A seguir estão detalhados os parâmetros do modelo.

n número total de jobs

m número de máquinas em série ($m = 2$)

p_i tempo de processamento da operação i

d_j prazo de entrega do job j

q_k quantidade de períodos de indisponibilidade na máquina k

u_{kl} instante de início do l-ésimo período de indisponibilidade na máquina k

e_{kl} instante de término do l-ésimo período de indisponibilidade na máquina k

M constante suficientemente grande

4.1.3 Variáveis de decisão

As variáveis de decisão traduzem as escolhas que o modelo deve realizar para construir um cronograma viável, como determinar inícios e términos das operações, identificar sequências dentro das máquinas e contabilizar atrasos. Por meio dessas variáveis, o modelo representa explicitamente as decisões de sequenciamento que impactam o desempenho final do sistema. A seguir estão detalhadas as variáveis de decisão do modelo.

s_i instante de início da operação i

c_i instante de término da operação i

U_i tempo adicional da operação i decorrente de períodos de indisponibilidade incidentes durante sua execução

x_{ik} variável binária que indica se a operação i é processada na máquina k

y_{ig} variável binária que indica a ordem relativa entre as operações i e g em uma mesma máquina

v_{ikl} variável binária que indica se a indisponibilidade l da máquina k ocorre antes do início da operação i

w_{ikl} variável binária que indica se a indisponibilidade l da máquina k ocorre antes do término da operação i

Ct_j tempo de conclusão do job j ao final da última máquina

T_j atraso do job j

4.2 Modelo de Programação Linear Inteira Mista (PLIM)

$$\min \sum_{j=1}^{n} T_j \tag{4.1}$$

Sujeito a:

$$\sum_{i=1}^{m} x_{\,i+\theta m,\,i} = m, \qquad \forall\, \theta = 0, 1, \ldots, n-1 \tag{4.2}$$

$$\sum_{i=1}^{nm} \sum_{k=1}^{m} x_{ik} = n\, m \tag{4.3}$$

$$\begin{aligned} s_{\,i+1+\theta m} &\geq s_{\,i+\theta m} + p_{\,i+\theta m} \\ s_{\,i+1+\theta m} &\geq c_{\,i+\theta m} \end{aligned} \qquad \forall\, i = 1, \ldots, m-1;\ \theta = 1, \ldots, n-1 \tag{4.4}$$

$$y_{ig} + y_{gi} \ \geq \ x_{ik} + x_{gk} - 1, \qquad \forall\, i, g = 1, \ldots, nm,\ i \neq g;\ k = 1, \ldots, m \tag{4.5}$$

$$s_g \ \geq \ c_i - (1 - y_{ig})\, M, \qquad \forall\, i, g = 1, \ldots, nm,\ i \neq g \tag{4.6}$$

$$\begin{aligned} v_{ikl} &\leq x_{ik} \\ s_i &\leq u_{kl} + M\, v_{ikl} + M\,(1 - x_{ik}) \\ s_i &\geq e_{kl} - M\,(1 - v_{ikl}) - M\,(1 - x_{ik}) \\ w_{ikl} &\leq x_{ik} \\ c_i &\leq u_{kl} + M\, w_{ikl} + M\,(1 - x_{ik}) \\ c_i &\geq e_{kl} + 1 - M\,(1 - w_{ikl}) - M\,(1 - x_{ik}) \end{aligned} \qquad \forall\, i = 1, \ldots, nm;\ k = 1, \ldots, m;\ l = 1, \ldots, q_k \tag{4.7}$$

$$U_i \ = \ \sum_{k=1}^{m} \sum_{l=1}^{q_k} (\, w_{ikl} - v_{ikl}\,)\,(e_{kl} - u_{kl}), \qquad \forall\, i = 1, \ldots, nm \tag{4.8}$$

$$c_i \ = \ s_i + p_i + U_i, \qquad \forall\, i = 1, \ldots, nm \tag{4.9}$$

$$Ct_j \;=\; c_{j,m}, \qquad \forall\, j = 1, \ldots, n \tag{4.10}$$

$$T_j \;\geq\; Ct_j - d_j, \qquad \forall\, j = 1, \ldots, n \tag{4.11}$$

$$T_j \;\geq\; 0, \qquad \forall\, j = 1, \ldots, n \tag{4.12}$$

A função objetivo apresentada na Equação (4.1) minimiza o atraso total dos jobs em relação às suas datas de entrega. Esse critério sintetiza o desempenho temporal do sistema sob indisponibilidades, capturando não apenas a eficiência global do cronograma, mas sobretudo a aderência a prazos individuais — elemento central em ambientes *make-to-order*.

As restrições (4.2) e (4.3) garantem a alocação estrutural correta das operações no ambiente de *flow shop*. A restrição (4.2) assegura que cada job seja processado em todas as máquinas do sistema, enquanto (4.3) impõe que exatamente $n \cdot m$ operações sejam atribuídas, mantendo a coerência entre o conjunto de operações modeladas e a configuração física do ambiente. Esse bloco estabelece a base combinatória do modelo, garantindo que toda solução candidata represente um sequenciamento completo e factível.

O conjunto de inequações em (4.4) representa a precedência tecnológica do *flow shop*: uma operação em uma máquina só pode iniciar após a operação correspondente na máquina imediatamente anterior ter sido concluída. Já a restrição (4.5) introduz a lógica de ordenação dentro de cada máquina, assegurando que quaisquer duas operações atribuídas ao mesmo recurso sejam sequenciadas por meio das variáveis binárias y_{ig}. Trata-se de um mecanismo clássico de imposição de precedência disjuntiva, fundamental para evitar sobreposição de processamento em um mesmo recurso.

A restrição (4.6) lineariza a relação de precedência entre pares de operações em uma mesma máquina por meio da técnica de Big-M. A variável binária y_{ig} determina se a operação i deve necessariamente preceder a operação g: quando $y_{ig} = 1$, o início de g só pode ocorrer após o término de i, o que é assegurado pela presença do termo c_i no lado direito da desigualdade. Quando $y_{ig} = 0$, o termo multiplicado por M relaxa a restrição, tornando-a automaticamente satisfeita. Esse mecanismo captura de forma precisa a lógica condicional do sequenciamento disjuntivo e garante que nenhum par de operações atribuídas à mesma máquina seja processado simultaneamente.

O bloco de restrições (4.7) trata explicitamente as janelas de indisponibilidade do tipo *resumable*. Essas desigualdades impedem que uma operação inicie ou termine dentro de um intervalo indisponível da máquina, e ainda evitam coincidências exatas com suas fronteiras. Com isso, o modelo captura corretamente o comportamento operacional esperado em paradas planejadas: interrupção imediata e retomada posterior, sem perda do progresso já realizado. Trata-se de um componente essencial para garantir a fidelidade do modelo ao ambiente real investigado.

A Equação (4.8) define o termo U_i, que representa o acréscimo temporal devido às indisponibilidades que ocorrem ao longo da execução da operação i. O par de variáveis binárias (v_{ikl}, w_{ikl}) identifica precisamente se o intervalo de downtime está contido no intervalo de processamento efetivo $[s_i, c_i]$. Dessa forma, cada ocorrência de indisponibilidade incidente sobre a operação gera um acréscimo igual à duração do intervalo, garantindo que o tempo modelado reflita a duração real do processamento sob paradas.

A restrição (4.9) vincula o início e o término das operações, agregando o tempo de processamento nominal e o tempo adicional U_i decorrente das indisponibilidades. Esse vínculo garante que o modelo contabilize de forma explícita o impacto das interrupções planejadas, refletindo a duração estendida do processamento.

As restrições (4.10)–(4.12) conectam o comportamento das operações ao desempenho final do sistema. A Equação (4.10) define o tempo de conclusão do job j como o término de sua última operação na máquina final do fluxo. Em seguida, (4.11) calcula o atraso individual a partir da diferenção entre essa conclusão e a respectiva data de entrega, enquanto (4.12) assegura que esse atraso seja não negativo. Juntas, essas restrições convertem o sequenciamento operacional em uma medida direta de qualidade temporal, alinhada ao objetivo de minimizar o atraso total.

5 MÉTODOS DE RESOLUÇÃO

O presente capítulo descreve, de forma detalhada, os métodos empregados na resolução do problema de *flow shop* permutacional com indisponibilidades do tipo retomável e critério de minimização do atraso total. Dada a natureza NP-difícil do problema e a inviabilidade prática de métodos exatos para instâncias de porte moderado, este capítulo apresenta a meta-heurística Variable Neighborhood Search (VNS) como abordagem principal adotada neste trabalho.

Inicialmente, expõe-se a lógica fundamental do VNS, destacando seus princípios teóricos, a estrutura de vizinhanças variáveis e o papel das etapas de *shaking*, busca local e processo decisório. Em seguida, o capítulo apresenta a estratégia de intensificação implementada por meio de uma busca local baseada exclusivamente no movimento *Relocate*, escolhido por sua capacidade de corrigir inversões locais relevantes para a redução do atraso total.

Posteriormente, o capítulo introduz os três operadores de *shaking* definidos para este estudo — *Swap*, *Chain Exchange* e *Downtime-Aware Relocate* — detalhando a função de cada um deles no processo de diversificação e mostrando como suas intensidades crescentes expandem progressivamente a região explorada do espaço de soluções. Em particular, o operador *Downtime-Aware Relocate* é descrito como uma adaptação específica ao problema estudado, uma vez que incorpora explicitamente as janelas de indisponibilidade das máquinas ao processo de perturbação, direcionando a busca para áreas mais promissoras.

Por fim, o capítulo apresenta a integração completa desses componentes no algoritmo VNS desenvolvido, descrevendo como cada operador é aplicado, como as soluções geradas são avaliadas e como a rotina de escalonamento — que reproduz fielmente o comportamento observado na formulação matemática — é utilizada para calcular o atraso total de cada sequência candidata. A seção final sintetiza o pseudocódigo completo do método, formalizando sua implementação e preparando o terreno para os experimentos computacionais apresentados no capítulo seguinte.

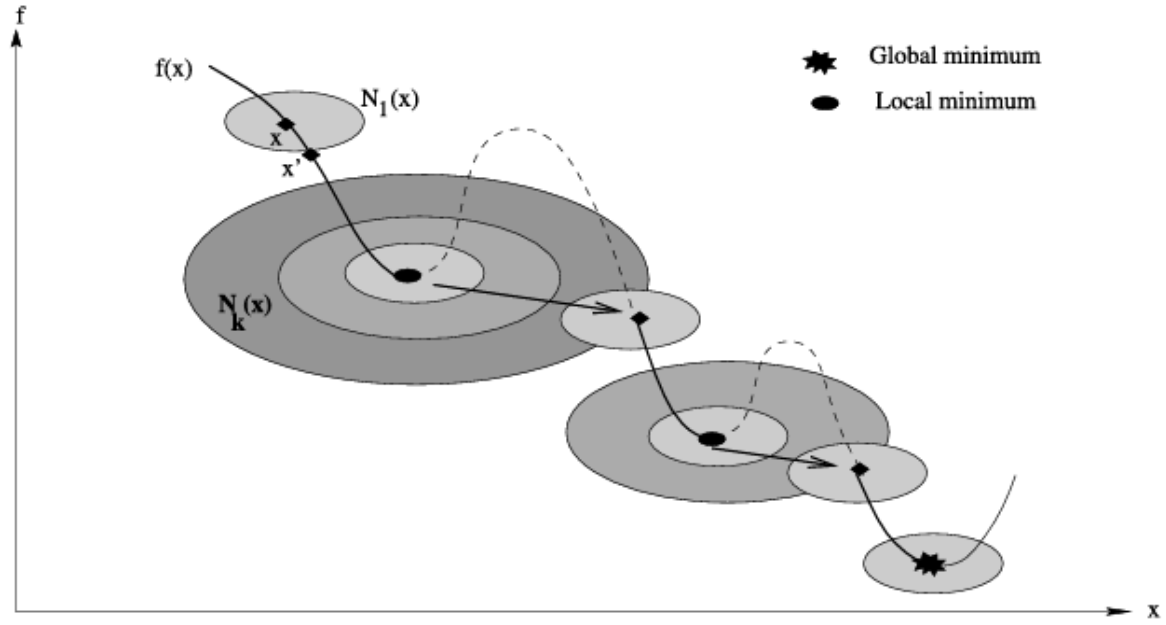
5.1 Estrutura Geral do Variable Neighbourhood Search (VNS)

O método *Variable Neighbourhood Search* (VNS), originalmente proposto por Mladenović e Hansen (1997), fundamenta-se na ideia de que a qualidade dos ótimos locais obtidos por heurísticas depende diretamente da estrutura de vizinhança sobre a qual esses ótimos são definidos. Como cada vizinhança enxerga apenas uma parte do espaço de soluções, um algoritmo restrito a uma única vizinhança tende a explorar repetidamente uma mesma região, limitando sua capacidade de encontrar soluções de melhor qualidade. Para justificar a exploração sistemática de múltiplas vizinhanças, os autores apresentam três fatos fundamentais que sustentam o método:

- um ótimo local em uma vizinhança dificilmente permanece ótimo quando avaliado sob outra vizinhança;
- o ótimo global é ótimo local em todas as vizinhanças;
- ótimos locais distintos tendem a localizar-se relativamente próximos entre si no espaço de soluções.

Esses fatos oferecem suporte teórico sólido para a alternância estruturada entre vizinhanças que caracteriza o VNS. Ao combinar perturbações de intensidades variadas com sucessivas aplicações de busca local, o método equilibra de forma natural os movimentos de diversificação — que ampliam o alcance da busca — e intensificação — que exploram em profundidade regiões que se mostram promissoras. A Figura 3 sintetiza visualmente essa dinâmica ao representar vizinhanças de diferentes alcances ao redor de uma solução corrente, explicitando como perturbações mais amplas permitem acessar regiões que não seriam exploradas por operadores mais restritos.

Figura 3: Representação geométrica do princípio central do *VNS*.



Fonte: extraído de Mladenović, Hansen e Pérez-Brito (2009).

A Figura 3, extraída de Mladenović, Hansen e Pérez-Brito (2009), desempenha um papel explicativo importante ao ilustrar o racional do método: cada conjunto representa uma vizinhança $N_k(x)$ em torno da solução x, e os pontos destacados simbolizam perturbações possíveis geradas na etapa de *shaking*. Após cada perturbação, uma busca local conduz a solução para uma região de menor valor da função objetivo dentro daquela vizinhança. Dessa forma, o diagrama traduz geometricamente o princípio fundamental do VNS: perturbar a solução o suficiente para acessar novas regiões e, em seguida, explorá-las intensivamente por meio da busca local.

Em sua formulação básica, o VNS estrutura-se em torno de três operações principais, que se sucedem de maneira sistemática ao longo da execução:

- **Perturbação (*Shaking*)**: responsável por gerar uma perturbação controlada da solução corrente, aplicando um operador pertencente à vizinhança N_k;
- **Busca local**: encarregada de refinar a solução perturbada, conduzindo-a a um ótimo local sob a vizinhança utilizada pela heurística de melhoria;
- **Processo decisório**: compara a solução obtida com a solução corrente e determina se o algoritmo deve reiniciar pela menor vizinhança ou avançar para uma vizinhança mais ampla.

Essa lógica operacional é formalizada no pseudocódigo clássico do método, apresentado no Algoritmo 1.

Algorithm 1 VNS Básico

```
1: function VNS(x, k_max, t_max)
2:     repeat
3:         k ← 1
4:         repeat
5:             x' ← Shake(x, k)
6:             x'' ← BuscaLocal(x')
7:             MudancaDeVizinhança(x, x'', k)
8:         until k = k_max
9:         t ← TempoCPU()
10:    until t > t_max
11: end function
```

A partir do pseudocódigo, observa-se que cada iteração inicia pela vizinhança mais curta, $k = 1$. Para cada vizinhança, aplica-se primeiramente o operador de *shaking*, produzindo uma solução intermediária x' obtida por meio de uma perturbação compatível com N_k. Essa etapa constitui o mecanismo de diversificação do método, deslocando a solução para uma região potencialmente distinta daquelas já exploradas.

Em seguida, x' é submetida à busca local, que produz x''. A busca local é aplicada de maneira sistemática, testando movimentos viáveis dentro da vizinhança utilizada por essa heurística até que nenhum movimento adicional produza melhoria. Assim, x'' representa um ótimo local para o operador da busca local, reforçando o papel dessa etapa como componente de intensificação: ela aprofunda a exploração da região visitada pelo *shaking* e identifica a melhor solução acessível dentro daquele subconjunto do espaço de busca.

Concluída a busca local, inicia-se o processo decisório. Se a solução x'' apresenta valor inferior ao da solução corrente, o método a aceita e reinicia a busca pela menor vizinhança ($k = 1$). Esse reinício permite explorar detalhadamente regiões recém-identificadas como promissoras. Caso contrário, a solução corrente é mantida e a vizinhança é ampliada ($k \leftarrow k + 1$), permitindo a aplicação de perturbações mais intensas quando perturbações curtas não são suficientes para alterar significativamente o estado da busca.

Esse ciclo de alternância entre perturbação, refinamento e expansão de vizinhança prossegue até que todas as vizinhanças sejam exploradas. Caso o critério de parada ainda não tenha sido atingido, um novo ciclo é iniciado com $k = 1$. Dessa forma, o VNS

combina de maneira disciplinada movimentos locais e movimentos amplos, adaptando-se dinamicamente à qualidade das soluções encontradas e mantendo uma trajetória de busca capaz de explorar regiões variadas do espaço de soluções de forma sistemática.

5.2 Busca local baseada no movimento *Relocate*

No contexto deste trabalho, a etapa de busca local do VNS tem como objetivo refinar a solução corrente, explorando modificações de pequena escala na sequência de jobs. A ideia central é examinar o entorno imediato da solução atual e identificar trocas locais que produzam redução na função objetivo, aqui representada pelo atraso total das tarefas. Dessa forma, a busca local atua como mecanismo de intensificação do método, concentrando esforços em regiões que já se mostraram promissoras após a etapa de *shaking*.

A busca local adotada é baseada no movimento *Relocate* (também conhecido na literatura como *insert move*). Em termos operacionais, esse movimento consiste em:

- remover um job da posição i da sequência corrente;
- inseri-lo em outra posição j da mesma sequência, com $i \neq j$, preservando a ordem relativa dos demais jobs.

Ao deslocar um único job para uma nova posição, o movimento *Relocate* permite corrigir inversões locais que podem estar contribuindo para atrasos elevados, com impacto relativamente limitado na estrutura global da sequência. Como o problema estudado envolve a minimização do atraso total, cada vizinho gerado por esse movimento é avaliado recalculando-se a função objetivo, considerando a dinâmica de processamento nas máquinas e as eventuais janelas de indisponibilidade.

A escolha do movimento *Relocate* como operador principal da busca local fundamenta-se em evidências consistentes da literatura sobre *flow shop* com critério de atraso total. Trabalhos clássicos e contemporâneos mostram que movimentos de inserção são particularmente eficazes para esse critério, produzindo reduções substanciais no *tardiness* ao reposicionar tarefas críticas em posições mais adequadas dentro da sequência.

Karabulut (2016) demonstra que heurísticas baseadas em inserção — tanto na construção inicial (NEHedd) quanto na fase de reconstrução e na busca local do algoritmo *Iterated Greedy* — desempenham papel central na obtenção de soluções de alta qualidade. De forma semelhante, Armentano e Ronconi (1999) utilizam a vizinhança de inserção

como mecanismo principal da busca tabu, indicando que esse movimento captura melhor os deslocamentos relevantes para minimizar atrasos. Vallada, Ruiz e Minella (2008) reforçam esse comportamento ao mostrar que heurísticas de melhoria baseadas em inserção superam sistematicamente heurísticas baseadas em troca simples (*swap*), especialmente quando o critério considerado é o atraso total.

Na prática, o movimento *Relocate* apresenta duas vantagens decisivas: (i) permite modificações estruturais significativas na sequência, reposicionando um job em qualquer posição e corrigindo diretamente inversões críticas associadas a datas de entrega; e (ii) mantém custo computacional moderado, podendo inclusive ser acelerado por técnicas de reutilização de cálculos, conforme sugerido em Karabulut (2016). Dessa forma, o operador oferece um equilíbrio favorável entre intensidade de exploração e custo computacional, sendo amplamente adotado em métodos de estado da arte para esse tipo de problema.

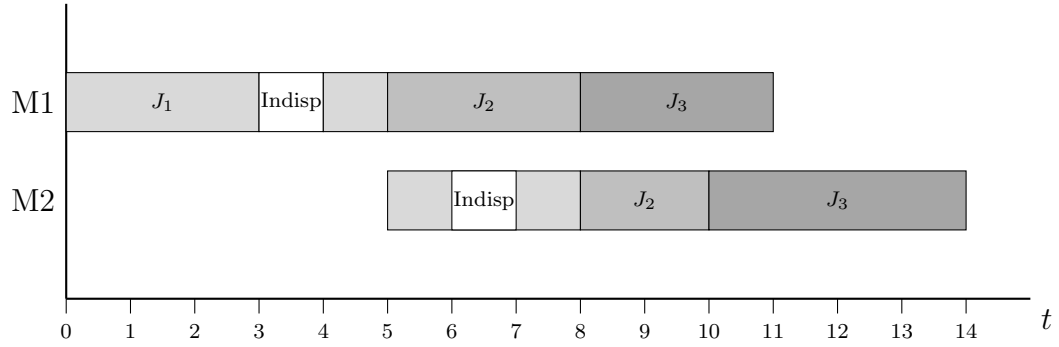
Essas evidências justificam a adoção do movimento *Relocate* como mecanismo exclusivo de intensificação na busca local deste trabalho, garantindo aderência às melhores práticas da literatura e contribuindo para a eficiência geral da meta-heurística desenvolvida.

Para ilustrar a lógica do movimento *Relocate*, considera-se uma instância hipotética com três jobs (J_1, J_2, J_3) e duas máquinas ($M1$ e $M2$), ambas sujeitas a janelas de indisponibilidade retomável. A sequência inicial $J_1 \rightarrow J_2 \rightarrow J_3$ resulta em um atraso total de 9 unidades, influenciado tanto pelos tempos de processamento quanto pelos atrasos decorrentes das interrupções.

Ao aplicar o movimento *Relocate*, o job J_3 é removido da terceira posição e reinserido na segunda, produzindo a nova sequência $J_1 \rightarrow J_3 \rightarrow J_2$. Essa realocação reduz o impacto da indisponibilidade de $M2$ sobre o job crítico J_3, resultando em atraso total de 6 unidades.

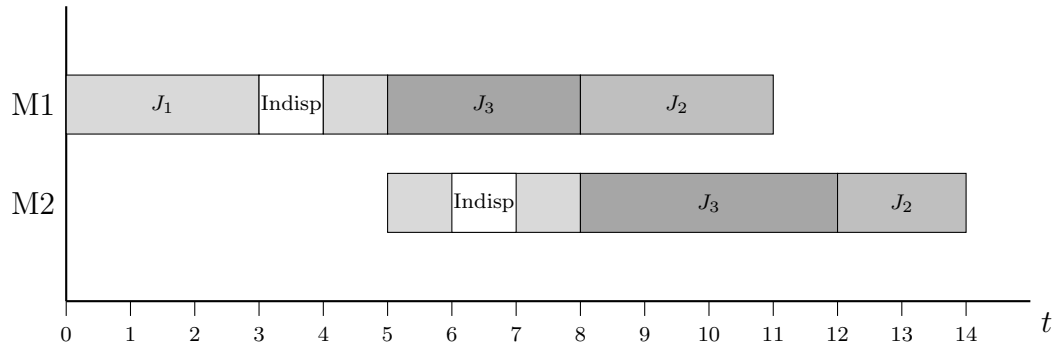
As Figuras 4 e 5 apresentam os diagramas de Gantt antes e depois da aplicação do movimento, permitindo visualizar claramente como a alteração da ordem modifica os tempos de processamento e os atrasos.

Figura 4: Gráfico de Gantt para a sequência $J_1 \rightarrow J_2 \rightarrow J_3$ antes da aplicação do movimento *Relocate*.



Fonte: autoria própria.

Figura 5: Gráfico de Gantt para a sequência $J_1 \rightarrow J_3 \rightarrow J_2$ após a aplicação do movimento *Relocate*.



Fonte: autoria própria.

O exemplo apresentado evidencia como o movimento *Relocate* pode alterar de forma significativa a dinâmica de processamento quando há janelas de indisponibilidade retomável. A simples realocação de um job para uma posição anterior na sequência modifica não apenas os horários de início e término, mas também a interação entre tarefas e períodos de indisponibilidade, impactando diretamente o atraso total da solução. Essa característica torna o movimento especialmente útil para corrigir inversões locais e reduzir atrasos acumulados. Motivado por essa capacidade de gerar melhorias estruturais com alterações pontuais na sequência, o movimento *Relocate* é adotado como base para o procedimento de intensificação utilizado neste trabalho. A seguir, detalha-se como esse movimento é sistematicamente explorado.

A lógica da busca local é do tipo melhoria iterativa. A partir de uma solução inicial x, o algoritmo gera todos os vizinhos obtidos pela aplicação do movimento *Relocate* em todas

as posições possíveis da sequência. Entre esses vizinhos, seleciona-se aquele que apresenta o menor valor de atraso total. Se essa solução for melhor do que a solução corrente, ela é aceita e passa a ser a nova solução de referência, reiniciando-se o processo de geração e avaliação de todos os movimentos *Relocate*. Caso nenhuma melhoria seja encontrada em relação à solução corrente, a busca local é encerrada, caracterizando a obtenção de um ótimo local com respeito ao movimento *Relocate*. O procedimento é sintetizado no Algoritmo 2.

Algorithm 2 Busca local baseada no movimento *Relocate*

```
1:  function BuscaLocal(x)
2:      calcular f(x)
3:      melhoria ← verdadeiro
4:      while melhoria do
5:          melhoria ← falso
6:          for all pares de posições (i, j) com i ≠ j do
7:              y ← Relocate(x, i, j)
8:              calcular f(y)
9:              if f(y) < f(x) then
10:                 x ← y
11:                 melhoria ← verdadeiro
12:                 break
13:             end if
14:         end for
15:     end while
16:     return x
17: end function
```

No Algoritmo 2, a condição de parada da busca local é satisfeita quando, ao término da avaliação de todos os movimentos *Relocate* possíveis, nenhuma solução com atraso total inferior ao da solução corrente é identificada. Nesse momento, considera-se que a busca atingiu um ótimo local com respeito à vizinhança induzida pelo movimento *Relocate*, e a solução resultante é devolvida ao procedimento principal do VNS. Essa estratégia garante que, antes de se recorrer novamente à etapa de *shaking*, a vizinhança imediata da solução corrente tenha sido explorada de forma exaustiva.

5.3 Movimentos de Perturbação

A etapa de *shaking* tem a função de diversificar a busca, gerando perturbações controladas na sequência corrente. Esses movimentos permitem ao algoritmo escapar do entorno imediato explorado pela busca local e acessar regiões distintas do espaço de soluções. Neste trabalho, definem-se três operadores de *shaking*: *Swap*, *Chain Exchange* e *Downtime-Aware Relocate*. Para facilitar a compreensão, cada movimento é acompanhado de um exemplo numérico simples para tangibilizar o seu funcionamento.

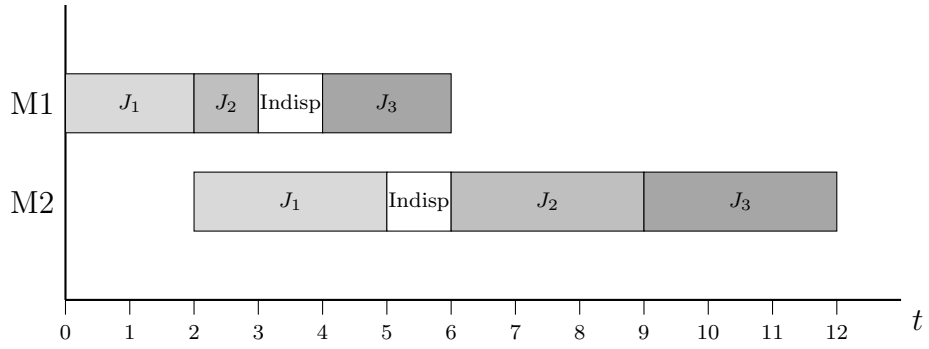
5.3.1 *Swap*

Para ilustrar o efeito do movimento *Swap*, considera-se uma instância hipotética composta por três jobs (J_1, J_2 e J_3) processados em duas máquinas ($M1$ e $M2$), ambas sujeitas a janelas de indisponibilidade retomável. Assume-se que $M1$ torna-se indisponível no intervalo $[3, 4]$ e $M2$ no intervalo $[5, 6]$. Os tempos de processamento dos jobs são iguais aos representados nos diagramas de Gantt, e as respectivas datas de entrega são: $d_1 = 8$, $d_2 = 12$ e $d_3 = 10$.

Na sequência inicial $J_1 \rightarrow J_2 \rightarrow J_3$, o job J_3 inicia seu processamento em $M2$ apenas após a passagem pela janela de indisponibilidade, concluindo no instante $C_3 = 12$. Como sua data de entrega é $d_3 = 10$, ele acumula dois períodos de atraso, resultando em $T_{\text{total}} = 2$ unidades.

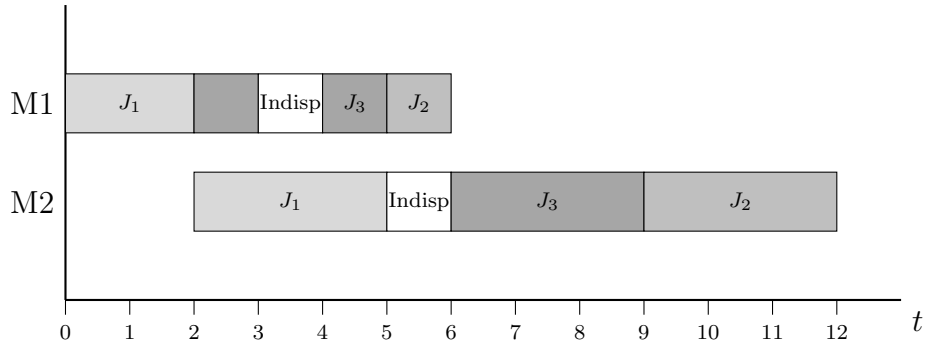
Aplicando o movimento *Swap* entre os jobs J_2 e J_3, obtém-se a sequência $J_1 \rightarrow J_3 \rightarrow J_2$. Nessa nova ordenação, J_3 é processado antes de J_2 e, consequentemente, inicia seu processamento em $M2$ imediatamente após a indisponibilidade, concluindo exatamente no instante $C_3 = 10$, dentro de sua data de entrega. O atraso total torna-se, portanto, $T_{\text{total}} = 0$. As Figuras 6 e 7 apresentam os diagramas de Gantt antes e depois do movimento, evidenciando como uma perturbação simples pode eliminar completamente o atraso associado ao job mais crítico.

Figura 6: Gráfico de Gantt para a sequência $J_1 \rightarrow J_2 \rightarrow J_3$ antes da aplicação de *Swap*.



Fonte: autoria própria.

Figura 7: Gráfico de Gantt para a sequência $J_1 \rightarrow J_3 \rightarrow J_2$ após a aplicação de *Swap*.



Fonte: autoria própria.

5.3.2 *Chain Exchange*

O movimento *Chain Exchange* seleciona um segmento contínuo da sequência de jobs e inverte a ordem dos elementos dentro desse segmento, substituindo o trecho original pela versão invertida. Trata-se de um operador de intensidade intermediária, análogo ao movimento *2-opt* amplamente utilizado em problemas de sequenciamento e roteamento. Por atuar sobre um bloco contínuo, o *Chain Exchange* permite reorganizar regiões inteiras da sequência, explorando configurações que o movimento *Swap* isolado não alcança, sem comprometer completamente a estrutura global da solução.

Para ilustrar o efeito desse movimento, considera-se uma instância com quatro jobs (J_1, J_2, J_3, J_4) processados em duas máquinas ($M1$ e $M2$), sujeitas a janelas de indisponibilidade retomável nos intervalos $[3, 4]$ em $M1$ e $[5, 6]$ em $M2$. Os tempos de processamento utilizados no exemplo são:

Tabela 2: Dados da instância hipotética com $m = 2$ e $n = 4$.

Trabalho	p_{1j}	p_{2j}	d_j
J_1	2	3	8
J_2	1	3	20
J_3	2	3	10
J_4	2	3	13

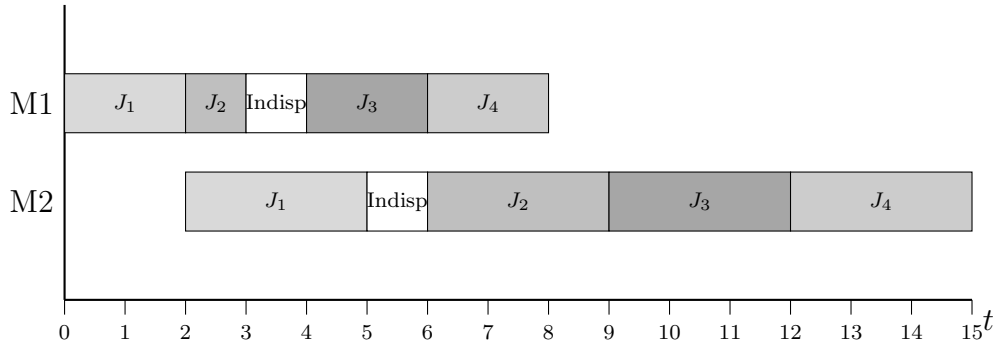
Fonte: autoria própria.

As datas de entrega consideradas são $d_1 = 8$, $d_2 = 20$, $d_3 = 10$ e $d_4 = 13$. Na sequência inicial $J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4$, os tempos de conclusão são $C_1 = 5$, $C_2 = 9$, $C_3 = 12$ e $C_4 = 15$. Assim, os jobs J_3 e J_4 apresentam atrasos de $T_3 = 2$ e $T_4 = 2$, resultando em atraso total $T_{\text{total}} = 4$. A Figura 8 apresenta o diagrama de Gantt correspondente.

Aplicando o movimento *Chain Exchange* ao segmento formado pelos jobs $[J_2, J_3, J_4]$, inverte-se a ordem do bloco, que passa a ser $[J_4, J_3, J_2]$. A nova sequência torna-se, portanto, $J_1 \rightarrow J_4 \rightarrow J_3 \rightarrow J_2$. Essa inversão desloca jobs originalmente atrasados para posições anteriores na sequência, reduzindo sua exposição às janelas de indisponibilidade e antecipando seus tempos de início em ambas as máquinas.

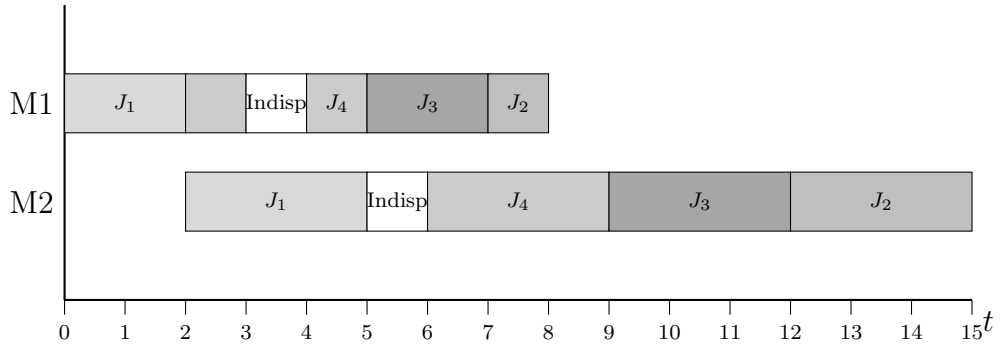
O novo escalonamento leva aos tempos de conclusão $C_1 = 5$, $C_4 = 9$, $C_3 = 12$ e $C_2 = 15$. Assim, J_4 deixa de incorrer em atraso, J_3 mantém atraso de dois períodos e J_1 e J_2 permanecem sem atraso. Dessa forma, o atraso total reduz-se para $T_{\text{total}} = 2$. A Figura 9 apresenta o diagrama de Gantt após a aplicação do movimento, evidenciando o impacto positivo da inversão da cadeia sobre a distribuição dos atrasos.

Figura 8: Gráfico de Gantt para a sequência $J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4$ antes da aplicação do movimento *Chain Exchange*.



Fonte: autoria própria.

Figura 9: Gráfico de Gantt para a sequência $J_1 \rightarrow J_4 \rightarrow J_3 \rightarrow J_2$ após a aplicação do movimento *Chain Exchange*.



Fonte: autoria própria.

5.3.3 *Downtime-Aware Relocate*

O movimento *Downtime-Aware Relocate* foi desenvolvido especificamente neste trabalho para lidar com instâncias do problema de *flow shop* que apresentam janelas de indisponibilidade retomável. Em ambientes desse tipo, quando um job inicia sua operação pouco antes de uma parada, seu processamento é interrompido e retomado mais tarde, prolongando o tempo de conclusão e podendo gerar atrasos significativos, que se propagam aos jobs subsequentes. O operador proposto explora essa estrutura temporal: ele identifica o job diretamente atravessado pela indisponibilidade em uma dada máquina e, sempre que possível, o troca de posição com o job imediatamente seguinte, desde que este consiga ser processado integralmente antes do downtime.

O *Downtime-Aware Relocate* identifica o job cuja operação seria diretamente interrompida pela janela de indisponibilidade e realiza a troca local desse job com o job imediatamente subsequente na sequência. Trata-se, portanto, de uma perturbação direcionada, que reposiciona o job mais prejudicado pelo downtime para uma posição imediatamente posterior à janela de parada, permitindo que seu processamento ocorra de forma contínua.

A ideia central é evitar o fracionamento desnecessário de operações causado pela indisponibilidade retomável. Ao inverter localmente a ordem entre o job afetado pela parada e o job seguinte, o operador redistribui a carga de processamento de modo a reduzir atrasos propagados ao longo da sequência. No exemplo apresentado, essa troca resulta em um efeito particularmente benéfico, pois o job subsequente consegue ser executado integralmente antes da janela de indisponibilidade, reforçando ainda mais o impacto positivo da perturbação. Em geral, no entanto, o *Downtime-Aware Relocate* aplica a troca indepen-

dentemente dessa condição, atuando sempre que identifica um job que seria atravessado pela janela de downtime.

Para ilustrar o funcionamento do operador, considera-se uma instância com quatro jobs (J_1, J_2, J_3 e J_4) processados em duas máquinas, $M1$ e $M2$. Ambas possuem janelas de indisponibilidade retomável, mas, neste exemplo, a decisão do movimento é tomada com base apenas na indisponibilidade de $M1$, no intervalo $[6, 8]$. A sequência inicial é

$$J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4.$$

Na máquina $M1$, J_1 e J_2 são processados sem interrupções, ocupando o intervalo de 0 a 4. O job J_3 inicia em $t = 4$ e teria duração de três unidades de tempo, de modo que sua operação atravessa a janela de indisponibilidade $[6, 8]$. Como o processamento é retomável, J_3 é executado parcialmente antes da parada (de 4 a 6) e retomado após o downtime, concluindo apenas em $t = 9$. O job seguinte, J_4, passa a iniciar em $t = 9$, terminando em $t = 11$. A Figura 10 mostra esse escalonamento, no qual o fracionamento de J_3 prolonga o tempo de fluxo e empurra J_4 para mais tarde.

Suponha que as datas de entrega sejam $d_1 = 8$, $d_2 = 10$, $d_3 = 20$ e $d_4 = 10$, e considere-se a máquina $M2$ como determinante para o cálculo do atraso. Na sequência inicial, os tempos de conclusão em $M2$ são $C_1 = 4$, $C_2 = 7$, $C_3 = 11$ e $C_4 = 13$, o que resulta em atraso total $T_{\text{total}} = 3$, concentrado em J_4, que termina acima de sua data de entrega ($C_4 = 13 > d_4 = 10$).

Aplicando-se o movimento *Downtime-Aware Relocate*, identifica-se que J_3 é o job atravessado pela indisponibilidade em $M1$, enquanto J_4 cabe integralmente no intervalo livre $[6, 8]$. O operador troca localmente a ordem desses jobs, produzindo a nova sequência

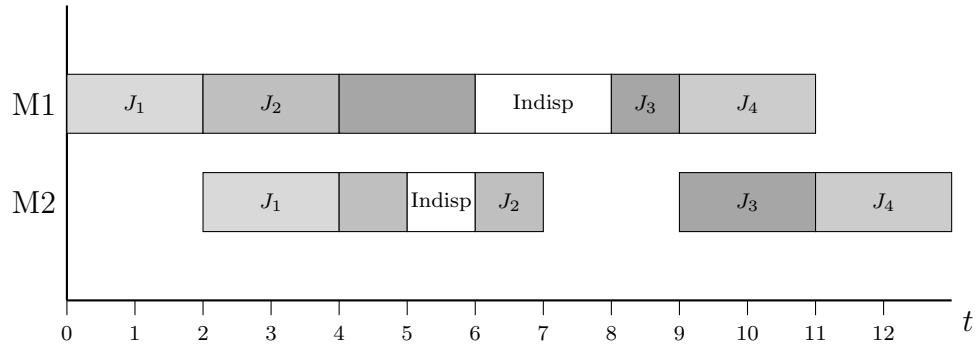
$$J_1 \rightarrow J_2 \rightarrow J_4 \rightarrow J_3.$$

Na máquina $M1$, J_4 passa a ser processado de forma contínua imediatamente antes da janela de downtime, ocupando o intervalo $[4, 6]$, enquanto J_3 é deslocado para após a parada, sendo executado de forma ininterrupta no intervalo $[8, 11]$. Em $M2$, os tempos de conclusão tornam-se $C_1 = 4$, $C_2 = 7$, $C_4 = 9$ e $C_3 = 13$. Todos os jobs passam a cumprir suas datas de entrega, resultando em atraso total $T_{\text{total}} = 0$. A Figura 11 apresenta o cronograma após o movimento, evidenciando a eliminação do fracionamento em $M1$ e a redução do atraso total.

Esse exemplo destaca o papel do *Downtime-Aware Relocate* como um movimento de

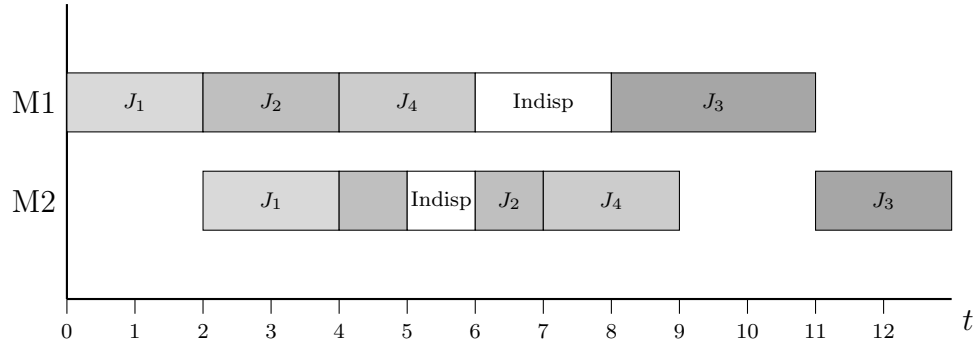
perturbação especificamente adaptado ao contexto de máquinas com indisponibilidades retomáveis. Ao incorporar informação temporal sobre as janelas de parada, o operador direciona a busca para regiões do espaço de soluções com maior potencial de redução de atraso.

Figura 10: Gráfico de Gantt para a sequência inicial $J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4$ antes da aplicação do movimento *Downtime-Aware Relocate*.



Fonte: autoria própria.

Figura 11: Gráfico de Gantt para a sequência $J_1 \rightarrow J_2 \rightarrow J_4 \rightarrow J_3$ após a aplicação do movimento *Downtime-Aware Relocate*.



Fonte: autoria própria.

5.4 Aplicação do VNS ao problema de *Flow Shop* com indisponibilidade de máquinas

A aplicação do método VNS ao problema estudado neste trabalho demanda adaptações específicas decorrentes das características estruturais do ambiente de produção e, em particular, da presença de janelas de indisponibilidade nas máquinas. Embora o VNS forneça uma estrutura geral baseada em perturbação, busca local e mudança de vizinhança, sua

implementação efetiva depende da forma como cada solução é avaliada, bem como da maneira como as restrições de disponibilidade influenciam o cálculo dos tempos de processamento e do atraso total.

No problema considerado, cada job deve ser processado em todas as máquinas seguindo a mesma ordem, característica típica do *flow shop*. As máquinas podem apresentar janelas de indisponibilidade ao longo do horizonte de planejamento, durante as quais nenhum processamento é permitido. Assim, para cada sequência candidata, o escalonamento deve ser recalculado a fim de determinar os instantes de início e término de cada operação, respeitando tanto a precedência entre máquinas quanto as restrições impostas pelas paradas. Esse recalculo é fundamental para medir corretamente o impacto da sequência na função objetivo.

O escalonamento utilizado segue três regras principais: (i) uma operação só pode ser iniciada quando a máquina estiver disponível e quando a operação anterior do job tiver sido concluída; (ii) caso o instante previsto de início ou término colida com uma janela de indisponibilidade, o processamento é automaticamente deslocado para o primeiro instante posterior em que a máquina esteja ativa; e (iii) ao final, o atraso de cada job é obtido comparando-se seu tempo de conclusão com sua data de entrega. Esse procedimento é aplicado para todas as soluções geradas pelo VNS, garantindo consistência na avaliação da função objetivo.

As operações definidas nas seções anteriores assumem papéis complementares dentro dessa estrutura. A busca local baseada no movimento *Relocate* atua como mecanismo de intensificação, explorando modificações de pequena escala na sequência com o objetivo de reduzir o atraso total. Antes que o algoritmo aplique perturbações mais amplas, essa etapa garante que a vizinhança imediata da solução corrente seja explorada de maneira exaustiva, conduzindo a solução a um ótimo local com respeito ao operador de realocação.

Os movimentos de *shaking*, por sua vez, permitem que o algoritmo escape desses ótimos locais. O operador *Swap* provoca pequenas perturbações ao trocar dois jobs da sequência. O movimento *Chain Exchange* altera a ordem de um segmento contínuo da sequência, produzindo perturbações de intensidade intermediária. Já o operador *Downtime-Aware Relocate* incorpora informações específicas sobre as janelas de indisponibilidade, realocando jobs cujo processamento seria interrompido por uma parada. Essa orientação operacional torna o método particularmente adequado ao problema, direcionando a busca para regiões com maior potencial de melhoria do atraso total.

A integração entre esses elementos segue a lógica tradicional do VNS: após a aplicação

de um operador de *shaking*, a solução obtida passa pela busca local. Caso apresente um valor inferior de atraso total em relação à solução corrente, a nova solução é aceita e a busca retorna para a primeira vizinhança. Caso contrário, o algoritmo avança para um operador de perturbação mais intenso. Esse mecanismo garante um equilíbrio entre exploração ampla do espaço de soluções e exploração detalhada de regiões promissoras, ajustando-se dinamicamente à qualidade das soluções encontradas.

A implementação também adota um critério de parada baseado em tempo máximo de execução, garantindo um compromisso adequado entre qualidade da solução e custo computacional. Em cada iteração, todas as vizinhanças são exploradas, assegurando que as perturbações definidas sejam utilizadas de forma consistente, independentemente das características da instância ou das janelas de indisponibilidade.

Em síntese, o método VNS foi adaptado para lidar explicitamente com a interação entre a ordem dos jobs, as janelas de indisponibilidade e o cálculo do atraso total. As vizinhanças foram definidas para oferecer diversidade suficiente de perturbações, enquanto a busca local foi estruturada para capturar melhorias pontuais de maneira eficiente. Essa combinação permite que o método navegue de forma estruturada no espaço de soluções, explorando alternativas de maneira coerente com as restrições operacionais do problema.

Algorithm 3 VNS para o problema de *flow shop* com indisponibilidade de máquinas

```
1: function VNS_FlowShop(x, k_max, t_max)
2:     definir N_1 = Swap, N_2 = Chain Exchange, N_3 = Downtime-Aware Relocate
3:     repeat
4:         k ← 1
5:         repeat
6:             x' ← Shake(x, N_k)
7:             x'' ← BuscaLocal_Relocate(x')
8:             if f(x'') < f(x) then
9:                 x ← x''
10:                k ← 1
11:            else
12:                k ← k + 1
13:            end if
14:        until k > k_max
15:        t ← TempoCPU()
16:    until t > t_max
17:    return x
18: end function
```

Em vista do algoritmo em questão, observa-se que o desempenho do VNS decorre da integração equilibrada entre seus mecanismos de perturbação e intensificação. Os três movimentos de *shaking* — *Swap*, *Chain Exchange* e *Downtime-Aware Relocate* — atuam de forma complementar, permitindo ao algoritmo escapar de ótimos locais de naturezas distintas. Por outro lado, a busca local baseada no movimento *Relocate* assegura que cada solução perturbada seja explorada de maneira rigorosa em seu entorno imediato, preservando a lógica de melhoria iterativa que caracteriza o VNS. Dessa forma, o método equilibra momentos de exploração ampla com fases de refinamento, conduzindo a busca de maneira estruturada pelo espaço de soluções. Essa estrutura consolida a base para a análise computacional desenvolvida no Capítulo 6, na qual o desempenho do VNS é avaliado sob instâncias de diferentes portes e padrões de indisponibilidade.

6 RESULTADOS E DISCUSSÃO

O presente capítulo apresenta e discute os experimentos computacionais realizados para avaliar o desempenho da meta-heurística VNS desenvolvida neste trabalho. Todos os procedimentos foram implementados em Python, utilizando códigos desenvolvidos especificamente para este projeto, incluindo rotinas de construção de soluções, operações de vizinhança, execução do VNS e simulação do processamento dos jobs sob janelas de indisponibilidade retomável.

Os experimentos foram executados em um Microcomputador Portátil Dell Inspiron 15 3530, equipado com processador *Intel Core i7-1355U*, *16 GB de memória RAM*, *SSD de 1 TB* e sistema operacional *Windows 11 Pro*. Esse ambiente computacional reflete uma configuração amplamente disponível em aplicações acadêmicas e industriais, permitindo avaliar o desempenho da meta-heurística em condições realistas, sem a necessidade de infraestrutura computacional dedicada ou servidores de alto desempenho.

A organização deste capítulo foi estruturada de modo a apresentar os resultados de forma progressiva e fundamentada. Inicialmente, analisa-se o impacto dos diferentes movimentos utilizados tanto na busca local quanto nas operações de *shaking*, discutindo como cada operador contribui para a redução do atraso total e para a capacidade de exploração do espaço de soluções. Em seguida, são apresentados os experimentos relativos às instâncias que possuem solução ótima conhecida, permitindo uma comparação direta entre os resultados do VNS e aqueles obtidos pelo modelo exato. Por fim, examina-se o comportamento da meta-heurística nas instâncias de médio e grande porte, nas quais o solver não alcança solução ótima dentro do tempo limite, avaliando a estabilidade do método, o crescimento do atraso total e o tempo necessário para execução.

Essa progressão permite conduzir a análise de forma sistemática, partindo da compreensão isolada dos operadores até a avaliação global do método em diferentes níveis de complexidade, oferecendo uma visão abrangente da qualidade e robustez do algoritmo proposto.

6.1 Comparação entre movimentos de perturbação

Dando início às análises computacionais, será avaliado de forma separada o desempenho de cada um dos movimentos que compõem a meta-heurística desenvolvida neste trabalho. Como o VNS incorpora tanto operadores de busca local quanto operadores de perturbação, compreender o comportamento individual de cada movimento é essencial para justificar a escolha do operador responsável pela intensificação da busca.

Para essa avaliação, utilizam-se as instâncias propostas por Dionizio (2024), já discutidas ao longo do trabalho e adequadas ao problema de *flow shop* com indisponibilidades retomáveis. Essas instâncias apresentam diversidade suficiente de tempos de processamento, datas de entrega e posicionamento das janelas de *downtime*, permitindo uma comparação robusta entre os movimentos considerados.

Quatro operadores são analisados de forma isolada: o movimento *Relocate*, utilizado originalmente como base da busca local, e os três operadores de perturbação empregados no procedimento de *shaking* — *Swap*, *Chain Exchange* e *Downtime-Aware Relocate*. Cada um deles altera a sequência de maneiras distintas e pode produzir efeitos diferentes tanto na redução do atraso total quanto na capacidade do algoritmo de escapar de ótimos locais.

O propósito desta etapa é identificar qual desses movimentos apresenta melhor desempenho quando utilizado como operador central de melhoria. A partir dessa comparação, torna-se possível avaliar se o *Relocate* representa, de fato, a estratégia mais eficiente de intensificação ou se algum dos demais operadores produz resultados superiores no contexto específico das instâncias testadas.

No experimento comparativo, cada movimento é avaliado como operador principal de busca local, substituindo temporariamente o Relocate na etapa de intensificação. A lógica de execução segue um procedimento de melhoria iterativa: dada uma solução inicial, o algoritmo aplica repetidamente o movimento selecionado em todas as posições possíveis da sequência, aceitando imediatamente qualquer solução vizinha que reduza o atraso total. Após cada melhoria, o processo é reiniciado a partir da nova solução, garantindo que o espaço local seja explorado de forma completa. A busca local é encerrada quando nenhum movimento adicional produz redução no valor da função objetivo, caracterizando a chegada a um ótimo local com respeito ao operador testado. A tabela apresentada a seguir detalha os resultados obtidos para cada movimento, permitindo comparar seus desempenhos e identificar aquele que apresenta o melhor comportamento para fins de intensificação.

Tabela 3: Resultados dos movimentos de busca local e perturbação para cada instância.

Inst.	n (jobs)	Relocate	Swap	Chain Exch.	DA-Relocate	Melhor mov.
1	5	360	362	369	367	Relocate
2	5	156	156	156	163	Swap
3	5	423	418	423	424	Swap
4	5	128	131	134	136	Relocate
5	5	556	560	557	557	Relocate
6	10	1063	1057	1060	1065	Swap
7	10	215	216	217	216	Relocate
8	10	618	618	612	618	ChainEx
9	10	295	291	290	285	DA
10	10	942	954	953	945	Relocate
11	15	499	414	323	506	ChainEx
12	15	349	349	353	360	Relocate
13	15	363	422	736	593	Relocate
14	15	248	247	414	166	DA
15	15	605	821	908	1073	Relocate
16	20	214	1074	301	358	Relocate
17	20	11	276	104	93	Relocate
18	20	316	242	37	595	ChainEx
19	20	793	748	53	664	ChainEx
20	20	322	715	39	721	ChainEx
21	25	466	690	1312	178	DA
22	25	34	12	45	30	ChainEx
23	25	44	260	540	1064	Relocate
24	25	1220	378	36	935	ChainEx
25	25	62	197	254	80	Relocate
26	30	325	291	345	1528	Swap
27	30	48	288	492	412	Relocate
28	30	348	533	828	33	DA
29	30	2	367	486	655	Relocate
30	30	85	625	391	160	Relocate
31	35	43	1133	1390	697	Relocate

Inst.	n (jobs)	Relocate	Swap	Chain Exch.	DA-Relocate	Melhor mov.
32	35	516	176	89	488	ChainEx
33	35	63	827	707	1115	Relocate
34	35	116	180	622	1576	Relocate
35	35	123	498	327	284	Relocate
36	40	1861	87	897	1334	Swap
37	40	964	164	728	745	Swap
38	40	1081	284	2087	751	Swap
39	40	1834	1527	871	456	DA
40	40	714	167	524	1089	Swap
41	45	219	1654	934	2192	Relocate
42	45	53	1393	1159	1314	Relocate
43	45	2923	58	2476	979	Swap
44	45	550	1146	4317	2750	Relocate
45	45	1204	248	1183	701	Swap
46	50	2537	60	2218	1207	Swap
47	50	54	1428	1404	1966	Relocate
48	50	309	560	1206	1548	Relocate
49	50	137	4734	3296	3688	Relocate
50	50	148	2094	1626	2439	Relocate
51	55	532	1134	2842	1585	Relocate
52	55	650	3828	916	2771	Relocate
53	55	1621	1227	2120	63	DA
54	55	339	5988	3960	6828	Relocate
55	55	131	3118	2093	2807	Relocate
56	60	355	1956	1606	1200	Relocate
57	60	137	284	1023	814	Swap
58	60	201	1572	3132	1201	Relocate
59	60	1188	2972	3714	4728	Relocate
60	60	108	924	1231	285	Relocate
61	65	256	2698	918	1570	Relocate
62	65	476	2638	2648	2666	Relocate
63	65	2099	38	392	1083	Swap
64	65	3361	6678	6917	6814	Relocate
65	65	315	2176	1985	524	Relocate

Inst.	n (jobs)	Relocate	Swap	Chain Exch.	DA-Relocate	Melhor mov.
66	70	185	918	2250	1229	Relocate
67	70	1350	3725	365	2413	ChainEx
68	70	538	3371	4028	2713	Relocate
69	70	7740	3058	2198	5721	ChainEx
70	70	394	2856	2966	1454	Relocate

Fonte: autoria própria.

A partir dos dados da Tabela, observa-se que o movimento *Relocate* apresenta o melhor desempenho em 41 das 70 instâncias analisadas, o que corresponde a aproximadamente 58,6% dos casos. Esse resultado indica que, de forma geral, o *Relocate* tende a produzir valores de atraso total menores com maior frequência, consolidando-se como o operador mais consistente entre os movimentos avaliados. Essa predominância ocorre em diferentes faixas de tamanho de instância, incluindo tanto problemas de pequeno porte quanto cenários de maior complexidade, o que sugere boa robustez do movimento quando utilizado como operador de intensificação na busca local.

Os demais movimentos também apresentam desempenhos competitivos em uma parcela relevante das instâncias. O operador *Swap* é responsável pelo melhor resultado em 13 instâncias (cerca de 18,6% do total), com destaque para alguns casos de menor porte e instâncias em que pequenas trocas pontuais entre jobs são suficientes para promover reduções expressivas no atraso total. O movimento *Chain Exchange* obtém o melhor desempenho em 10 instâncias (aproximadamente 14,3%), indicando que, em certos casos, a realocação simultânea de cadeias de jobs permite explorar regiões do espaço de soluções que não são facilmente acessadas por movimentos mais simples. Já o *Downtime-Aware Relocate* aparece como melhor operador em 6 instâncias (cerca de 8,6%), tipicamente em cenários específicos em que o posicionamento das janelas de indisponibilidade exerce papel mais determinante na qualidade final da solução.

De maneira geral, a distribuição dos melhores resultados mostra que, embora haja instâncias em que *Swap*, *Chain Exchange* ou *Downtime-Aware Relocate* sejam superiores, o *Relocate* permanece como o movimento com melhor desempenho médio e maior frequência de vitórias. Esse comportamento reforça a decisão de adotá-lo como operador central da etapa de busca local no VNS proposto, enquanto os demais movimentos são mantidos na fase de perturbação, desempenhando um papel complementar de diversificação do processo de busca.

6.2 VNS para instâncias com solução ótima conhecida

Dando continuidade à avaliação computacional, esta subseção apresenta a comparação entre os resultados obtidos pelo método VNS e as soluções ótimas disponíveis para o conjunto de instâncias de menor porte. Essas instâncias correspondem às dez primeiras do conjunto de dados utilizado, todas com solução ótima previamente identificada pelo modelo matemático. Por se tratar de problemas de baixa complexidade relativa, o tempo máximo de execução do VNS foi limitado a 300 segundos (5 minutos). Esse limite é suficiente para permitir que a meta-heurística explore adequadamente o espaço local dessas instâncias menores, ao mesmo tempo em que estabelece uma base comparativa para análises posteriores. No capítulo seguinte, ao tratar de instâncias de maior porte, esse limite será revisado de forma a permitir uma discussão mais ampla sobre como o aumento do tempo de execução impacta a qualidade das soluções encontradas.

A Tabela 4 apresenta os resultados obtidos. Para cada instância, são reportados o número de jobs, o valor ótimo conhecido e a solução retornada pelo VNS, juntamente com o *gap* percentual entre esses valores.

Tabela 4: Comparação entre o VNS e a solução ótima nas instâncias pequenas.

Inst.	n (jobs)	Valor ótimo	Valor VNS	Gap (%)
1	5	359	380	6.09
2	5	154	165	7.60
3	5	417	433	4.00
4	5	128	135	5.51
5	5	554	580	4.73
6	10	1057	1104	4.46
7	10	214	224	4.93
8	10	610	644	5.73
9	10	285	302	5.98
10	10	942	1005	6.69

Fonte: autoria própria.

Os resultados da Tabela 4 mostram que o método VNS apresenta desempenho bas-

tante consistente ao comparar suas soluções com os valores ótimos conhecidos. Observa-se que o *gap* médio permanece significativamente abaixo de 10%, concentrando-se principalmente na faixa entre 4% e 7%. Esse comportamento demonstra que, mesmo com um tempo de execução limitado, o algoritmo é capaz de explorar de forma eficiente o espaço de soluções dessas instâncias pequenas, retornando resultados próximos do ótimo em todas as situações analisadas.

O desempenho uniforme ao longo das diferentes instâncias reforça a robustez do VNS quando aplicado a cenários de menor porte. A baixa variabilidade entre os *gaps* sugere que o processo de intensificação e as estratégias de vizinhança adotadas são eficazes na condução da busca para regiões de alta qualidade da solução. Além disso, o fato de todas as soluções retornadas permanecerem relativamente próximas do ótimo indica que a meta-heurística não sofre de instabilidade ou degradação súbita em termos de qualidade, mesmo considerando diferentes estruturas de tempo de processamento e janelas de indisponibilidade.

Esses resultados servem como base sólida para as análises subsequentes com instâncias maiores, nas quais o valor ótimo não está mais disponível. Nessas situações, o tempo de execução será ampliado para avaliar mais detalhadamente como o VNS se comporta quando enfrenta instâncias mais complexas e como o aumento de tempo permitido influencia a qualidade final das soluções.

6.3 VNS para instâncias sem solução ótima conhecida

Para as demais instâncias do conjunto de testes, não há informação sobre o valor ótimo da função objetivo, uma vez que o modelo exato não foi capaz de encontrar a solução ótima dentro do limite de tempo estipulado. Ainda assim, o método exato fornece, para todas essas instâncias, um *upper bound* associado à melhor solução viável identificada. Nesta subseção, o desempenho do VNS é avaliado justamente em relação a esse *upper bound*, permitindo quantificar o quão próximas as soluções heurísticas se encontram das melhores soluções conhecidas.

Consideram-se, aqui, as 60 instâncias restantes (da 11 até a 70), abrangendo problemas de porte intermediário e grande. Para cada instância, o VNS foi executado com três limites de tempo distintos: 300 segundos (5 minutos), 600 segundos (10 minutos) e 900 segundos (15 minutos). Em cada um desses limites, registra-se o melhor valor de atraso

total encontrado até o término da execução, bem como o *gap* percentual em relação ao *upper bound* gerado pelo método exato. A Tabela 5 resume esses resultados.

Tabela 5: Comparação entre o VNS e o *upper bound* nas instâncias sem solução ótima conhecida.

Inst.	n (jobs)	UB	VNS 5 min	Gap 5 min (%)	VNS 10 min	Gap 10 min (%)	VNS 15 min	Gap 15 min (%)
11	15	1828	2128	16.41	2058	12.57	2042	11.70
12	15	358	416	16.09	399	11.51	390	8.83
13	15	992	1121	13.00	1103	11.18	1101	11.02
14	15	644	744	15.53	737	14.41	728	13.04
15	15	1664	1950	17.19	1915	15.08	1881	13.05
16	20	214	247	15.42	243	13.60	241	12.62
17	20	11	13	18.86	12	15.42	12	14.55
18	20	316	372	17.86	358	13.19	352	11.39
19	20	793	901	13.61	886	11.70	877	10.58
20	20	322	366	13.58	360	11.86	358	11.38
21	25	466	542	16.25	529	13.50	520	11.58
22	25	34	40	18.24	38	13.74	37	11.75
23	25	44	50	15.68	49	12.86	49	11.58
24	25	1220	1408	15.38	1385	13.52	1370	12.25
25	25	62	71	14.60	69	11.16	68	9.22
26	30	325	387	18.93	374	15.01	370	13.89
27	30	48	56	16.92	53	11.21	52	9.25
28	30	348	395	13.64	384	10.28	380	9.09
29	30	2	2	16.25	2	14.46	2	14.12
30	30	85	96	12.75	94	10.95	93	9.97
31	35	43	49	14.37	47	11.48	47	10.15
32	35	516	607	17.68	589	14.19	581	12.68
33	35	63	74	17.67	72	14.20	71	13.02
34	35	116	132	13.64	129	11.38	127	9.78

Inst.	n (jobs)	UB	VNS 5 min	Gap 5 min (%)	VNS 10 min	Gap 10 min (%)	VNS 15 min	Gap 15 min (%)
35	35	123	146	18.37	142	15.42	140	14.06
36	40	1861	2196	18.03	2119	13.86	2091	12.35
37	40	964	1150	19.33	1109	14.97	1093	13.35
38	40	1081	1247	15.39	1214	12.29	1199	10.85
39	40	1834	2168	18.19	2104	14.71	2076	13.22
40	40	714	842	18.02	810	13.47	795	11.41
41	45	219	253	15.38	246	12.40	243	11.13
42	45	53	62	16.24	60	13.29	59	11.87
43	45	2923	3397	16.24	3319	13.56	3278	12.14
44	45	550	655	19.09	631	14.66	622	13.09
45	45	1204	1407	16.86	1357	12.75	1338	11.17
46	50	2537	2957	16.53	2890	13.90	2864	12.91
47	50	54	64	17.63	61	13.99	60	11.78
48	50	309	362	17.28	349	13.00	342	10.59
49	50	137	161	17.53	156	13.84	154	12.21
50	50	148	177	19.53	170	14.77	168	13.25
51	55	532	617	15.93	603	13.20	595	11.79
52	55	650	759	16.75	735	13.20	726	11.70
53	55	1621	1911	17.91	1859	14.65	1827	12.74
54	55	339	401	18.19	385	13.44	379	11.78
55	55	131	153	16.86	147	12.59	145	11.17
56	60	355	414	16.65	399	12.42	391	10.18
57	60	137	159	15.98	154	12.57	152	11.11
58	60	201	234	16.37	227	12.83	224	11.32
59	60	1188	1379	16.10	1329	11.86	1314	10.54
60	60	108	127	17.87	121	12.04	119	10.62
61	65	256	296	15.78	289	12.93	284	10.95
62	65	476	558	17.32	541	13.68	532	11.78
63	65	2099	2442	16.37	2363	12.60	2338	11.39

Inst.	n (jobs)	UB	VNS 5 min	Gap 5 min (%)	VNS 10 min	Gap 10 min (%)	VNS 15 min	Gap 15 min (%)
64	65	3361	3977	18.31	3821	13.67	3754	11.68
65	65	315	367	16.67	355	12.53	349	10.80
66	70	185	214	15.79	209	13.21	207	12.00
67	70	1350	1579	16.99	1523	12.83	1499	11.04
68	70	538	627	16.68	611	13.46	603	12.08
69	70	7740	9089	17.42	8754	13.12	8628	11.47
70	70	5052	5835	15.49	5731	13.44	5712	13.07

Fonte: autoria própria.

A análise dos resultados da Tabela 5 mostra que, nas instâncias em que o valor ótimo não é conhecido, o VNS mantém um desempenho satisfatório em relação ao *upper bound* fornecido pelo modelo exato. Em média, o *gap* entre a melhor solução heurística e o *upper bound* situa-se em torno de 16% para o limite de 5 minutos, reduzindo-se para aproximadamente 13% com 10 minutos de execução e para cerca de 12% quando o tempo é ampliado para 15 minutos. Esse comportamento indica que o aumento do tempo computacional contribui para a melhoria da qualidade das soluções, mas que os ganhos marginais tendem a diminuir à medida que o tempo cresce.

Outro aspecto relevante é que, embora haja uma melhora consistente entre 5 e 10 minutos para praticamente todas as instâncias, a redução adicional de *gap* entre 10 e 15 minutos é, em muitos casos, relativamente pequena. Esse efeito torna-se particularmente evidente nas instâncias de maior porte, com 60, 65 e 70 jobs, nas quais a diferença entre as soluções encontradas em 10 e 15 minutos é frequentemente inferior a dois pontos percentuais. Isso sugere que, a partir de determinado patamar de tempo de execução, o VNS tende a se estabilizar em uma região de boa qualidade do espaço de soluções, encontrando dificuldades para escapar de vales locais mais profundos mesmo com um aumento moderado no tempo disponível.

Um ponto que merece destaque é que todo esse comportamento é obtido com a calibração de um único parâmetro: o tempo máximo de execução do algoritmo. Diferentemente de outras heurísticas e meta-heurísticas que dependem da escolha simultânea de diversos parâmetros (como tamanhos de vizinhança, probabilidades de aceitação, taxas de resfriamento ou pesos em funções de avaliação), o VNS proposto exige apenas a definição

do horizonte de tempo disponível para a busca. Na prática, isso simplifica de forma significativa o processo de configuração do método, reduz o esforço de calibração e torna o algoritmo mais atraente para aplicações reais, nas quais o usuário pode simplesmente ajustar o tempo de execução de acordo com a disponibilidade computacional e o nível de aproximação desejado em relação ao *upper bound*.

De maneira geral, os resultados obtidos ao longo desta seção demonstram que o VNS desenvolvido apresenta desempenho consistente e adequado para o problema de *flow shop* com indisponibilidades retomáveis. Nos testes em que a solução ótima é conhecida, o método mostrou-se capaz de produzir resultados muito próximos do ótimo mesmo com um tempo restrito de execução. Nas instâncias de porte intermediário e grande, nas quais o valor ótimo não está disponível, o VNS manteve *gaps* controlados em relação ao *upper bound*, apresentando melhora gradual com o aumento do tempo computacional e estabilização dos ganhos em horizontes mais longos. Além disso, a simplicidade do processo de configuração — baseado exclusivamente no ajuste do tempo máximo de execução — reforça o potencial de aplicação prática do método, já que o usuário pode calibrar facilmente o equilíbrio entre qualidade da solução e custo computacional. Esses achados consolidam o VNS como uma estratégia heurística eficiente, estável e de baixo esforço de parametrização, criando uma base sólida para a discussão final e as conclusões do trabalho.

7 CONCLUSÃO

O presente trabalho investigou um problema de *flow shop* permutacional com duas máquinas, envolvendo tarefas retomáveis, janelas determinísticas de indisponibilidade e minimização do atraso total — um conjunto de características que, quando combinadas, ampliam significativamente a complexidade estrutural do problema e o distanciam dos modelos clássicos amplamente discutidos na literatura. A formulação matemática reproduzida a partir de Dionizio (2024) forneceu a base conceitual necessária para compreender com rigor a dinâmica do processamento sob indisponibilidades retomáveis e para validar experimentalmente as soluções encontradas pela abordagem heurística.

Ao longo do desenvolvimento do texto, evidenciou-se que a introdução simultânea de critérios de atraso, comportamento retomável e paradas programadas cria um ambiente combinatório altamente desafiador. A revisão bibliográfica mostrou que, embora haja estudos sólidos sobre *flow shop* com *tardiness* e outros sobre indisponibilidade de máquinas, a interseção entre esses temas permanece pouco explorada, com apenas dois trabalhos dedicados especificamente ao caso retomável com minimização do atraso total. Nesse contexto, levantamentos abrangentes como os de Vallada & Ruiz (2010), Karabulut (2016), Berrais et al. (2014) e Rakrouki et al. (2022) demonstram que métodos clássicos e metaheurísticas populacionais obtiveram avanços importantes, mas ainda não contemplavam uma abordagem baseada em VNS focada nesse recorte específico.

A formulação matemática analisada neste trabalho permitiu uma compreensão profunda das restrições operacionais envolvidas, especialmente no que diz respeito ao tratamento das janelas de indisponibilidade e da modelagem da retomada das operações. A estrutura de precedência entre máquinas, as relações disjuntivas internas a cada recurso e o cálculo preciso do tempo adicional causado pelas paradas (U_i) mostraram-se essenciais para representar adequadamente o ambiente produtivo e fundamentar os mecanismos de avaliação utilizados pela meta-heurística implementada.

Com base nesses elementos teóricos e estruturais, foi desenvolvida uma versão adaptada do *Variable Neighborhood Search* (VNS), construída de modo a refletir tanto as me-

lhores práticas da literatura quanto as especificidades do problema estudado. A definição das vizinhanças — *Swap*, *Chain Exchange* e o operador especializado *Downtime-Aware Relocate* — buscou equilibrar intensificação e diversificação da busca. A escolha do movimento *Relocate* como operador exclusivo da busca local foi confirmada empiricamente pelos experimentos da Seção 6.1, que demonstraram seu desempenho superior em aproximadamente 60% das instâncias avaliadas, em consonância com achados consolidados relacionados ao critério de atraso total.

Os resultados computacionais demonstraram a eficiência da abordagem proposta. Para as instâncias de pequeno porte, em que a solução ótima era conhecida, o VNS apresentou *gaps* extremamente reduzidos, geralmente entre 4% e 7%, confirmando sua capacidade de se aproximar de soluções ótimas em tempos de execução bastante razoáveis. Esse desempenho reforça que a meta-heurística, mesmo operando de forma relativamente simples, com poucos parâmetros e estrutura clara, possui forte poder de intensificação e boa estabilidade.

Para as instâncias de porte intermediário e grande, nas quais apenas *upper bounds* eram conhecidos, a análise revelou *gaps* consistentemente controlados, próximos de 16% com 5 minutos de execução e reduzidos para aproximadamente 12% quando o tempo era ampliado para 15 minutos. Observou-se uma melhora clara entre 5 e 10 minutos de execução, seguida de ganhos marginais entre 10 e 15 minutos, indicando que o algoritmo converge de forma relativamente rápida para regiões de soluções robustas. Esse comportamento ratifica a adequação do VNS para cenários em que há limitação de tempo de processamento — condição comum em aplicações industriais — ao mesmo tempo em que evidencia o bom equilíbrio entre profundidade de busca e custo computacional.

Um aspecto de destaque nos resultados é que o único parâmetro relevante do algoritmo é o tempo de execução permitido. Essa característica torna a abordagem altamente aplicável em contextos reais, onde não há disponibilidade para longas rotinas de calibração ou ajuste fino de múltiplos parâmetros. O usuário pode simplesmente ajustar o tempo conforme a necessidade de qualidade da solução ou a disponibilidade computacional, tornando a meta-heurística prática, eficiente e de fácil implementação.

Em síntese, o trabalho apresentou contribuições conceituais, metodológicas e computacionais relevantes. Conceitualmente, consolidou um recorte de problema ainda pouco explorado na literatura. Metodologicamente, desenvolveu uma meta-heurística alinhada a esse recorte, combinando operadores tradicionais com adaptações específicas para o tratamento das indisponibilidades. Computacionalmente, demonstrou que a abordagem

VNS é capaz de oferecer soluções de boa qualidade, mantendo simplicidade operacional e tempos de execução razoáveis.

Como direções para pesquisas futuras, destacam-se três caminhos naturais: (i) a extensão do modelo para ambientes com três ou mais máquinas, onde a complexidade cresce consideravelmente; (ii) o desenvolvimento de versões híbridas do VNS incorporando ideias de algoritmos genéticos, *Iterated Greedy* ou *Simulated Annealing*, de modo a ampliar a diversidade global da busca; e (iii) a análise de cenários estocásticos ou com múltiplas janelas de indisponibilidade irregulares, aproximando ainda mais o problema de condições reais encontradas na indústria.

Por fim, reforça-se que o objetivo proposto de demonstrar a viabilidade e eficácia do VNS para o *flow shop* permutacional com indisponibilidades retomáveis e critério de atraso total foi plenamente alcançado. O método mostrou-se capaz de navegar com eficiência em um espaço de busca altamente complexo, fornecendo soluções competitivas e estabelecendo uma base sólida para investigações futuras sobre esse tema desafiador e de grande relevância prática.

REFERÊNCIAS

ALAOUI, H.; ARTIBA, A.; ELMAGHRABY, S. E.; RIANE, F. Scheduling of a two-machine flowshop with availability constraints on the first machine. [S.l.: s.n.], 2005.

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

BERRAIS, A.; RAKROUKI, M. A.; LADHARI, T. Minimizing total tardiness in two-machine permutation flowshop problem with availability constraints and subject to release dates. In: INTERNATIONAL CONFERENCE ON PROJECT MANAGEMENT AND SCHEDULING (PMS), 14., 2014, Munich. *Proceedings*. Munich: TUM School of Management, 2014. p. 32.

BLAZEWICZ, J.; BREIT, J.; FORMANOWICZ, P.; KUBIAK, W.; SCHMIDT, G. Heuristic algorithms for the two-machine flowshop with limited machine availability. *Omega*, v. 29, 2001.

BREIT, J. An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint. *Information Processing Letters*, v. 90, 2004.

CHENG, T. C. E.; WANG, G. An improved heuristic for two-machine flowshop scheduling with an availability constraint. *Operations Research Letters*, v. 26, 2000.

DIONÍZIO, L.; RONCONI, D. Desenvolvimento de um modelo de programação inteira mista para ambiente de flow shop resumable. In: SIMPÓSIO DE ENGENHARIA DE PRODUÇÃO (SIMPEP), 2024.

FERNANDEZ-VIAGAS, V.; VALENTE, J. M. S.; FRAMINAN, J. M. Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. *Expert Systems with Applications*, 2017.

KARABULUT, K. A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Computers & Industrial Engineering*, 2016.

KIM, Y.-D. Heuristics for flowshop scheduling problems: minimizing mean tardiness. [S.l.]: Operational Research Society Ltd, 1993.

KOULAMAS, C. The proportionate flow shop total tardiness problem. *European Journal of Operational Research*, 2020.

LEE, C.-Y. Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint. *Operations Research Letters*, v. 20, 1996.

LEE, J.-Y.; KIM, Y.-D. Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraint on the first machine. *Operations Research Letters*, v. 20, 1997.

LENSTRA, J. K.; RINNOOY KAN, A. H. G.; BRUCKER, P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, v. 1, p. 343–362, 1977.

MA, Y.; CHU, C.; ZUO, C. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 2009.

MLADENOVIC, N.; HANSEN, E. Variable neighborhood search. Montreal: GERAD, 1997.

PINEDO, M. L. *Scheduling: Theory, Algorithms, and Systems*. 5. ed. [S.l.]: Springer, 2016.

RAKROUKI, M. A.; ALJOHANI, A.; ALHARBE, N.; BERRAIS, A.; LADHARI, T. Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraints. *Intelligent Automation & Soft Computing*, 2023. DOI: 10.32604/iasc.2023.028604.

SCHALLER, J. Note on minimizing total tardiness in a two-machine flowshop. *Computers & Operations Research*, v. 32, 2004.

SCHMIDT, G. Scheduling with limited machine availability. *European Journal of Operational Research*, 2014.

VALLADA, E.; RUIZ, R.; MINELLA, G. Minimising total tardiness in the m-machine flowshop problem: a review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 2006.

VALLADA, E.; RUIZ, R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, 2010.

WANG, X.; CHENG, T. C. E. An approximation scheme for two-machine flowshop scheduling with setup times and an availability constraint. *Computers & Operations Research*, v. 34, p. 2894–2901, 2005.

ZHANG, X.; CHEN, L. A general variable neighborhood search algorithm for a parallel-machine scheduling problem considering machine health conditions and preventive maintenance. *Computers & Operations Research*, v. 143, 105738, 2022.