UNIVERSIDADE FEDERAL DE VIÇOSA

Métodos Híbridos de Otimização Combinatória e Aprendizado por Reforço para Problemas Integrados de Produção e Distribuição

Matheus de Freitas Araújo
Doctor Scientiae

VIÇOSA - MINAS GERAIS
2024

MATHEUS DE FREITAS ARAÚJO

Métodos Híbridos de Otimização Combinatória e Aprendizado por Reforço para Problemas Integrados de Produção e Distribuição

Tese apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Doctor Scientiae*.

Orientador: Thiago Henrique Nogueira

Coorientador: Jose Elias Claudio Arroyo

VIÇOSA - MINAS GERAIS
2024

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
A663m
2024

Araujo, Matheus de Freitas, 1991-
Métodos híbridos de otimização combinatória e aprendizado por reforço para problemas integrados de produção e distribuição / Matheus de Freitas Araujo. – Viçosa, MG, 2024.
1 tese eletrônica (185 f.): il. (algumas color.).

Inclui apêndice.
Orientador: Thiago Henrique Nogueira.
Tese (doutorado) - Universidade Federal de Viçosa, Departamento de Informática, 2024.
Referências bibliográficas: f. 120-127.
DOI: https://doi.org/10.47328/ufvbbt.2025.012
Modo de acesso: World Wide Web.

1. Programação heurística. 2. Programação linear. 3. Planejamento da produção. 4. Aprendizado do computador. I. Nogueira, Thiago Henrique, 1982-. II. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 22. ed. 005.1

Bibliotecário(a) responsável: Euzébio Luiz Pinto CRB-6/3317

MATHEUS DE FREITAS ARAÚJO

Métodos Híbridos de Otimização Combinatória e Aprendizado por Reforço para Problemas Integrados de Produção e Distribuição

Tese apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Doctor Scientiae*.

APROVADA: 9 de outubro de 2024.

Assentimento:

_____

Matheus de Freitas Araújo
Autor

_____

Thiago Henrique Nogueira
Orientador

Essa tese foi assinada digitalmente pelo autor em 09/01/2025 às 12:14:17 e pelo orientador em 09/01/2025 às 13:39:37. As assinaturas têm validade legal, conforme o disposto na Medida Provisória 2.200-2/2001 e na Resolução nº 37/2012 do CONARQ. Para conferir a autenticidade, acesse https://siadoc.ufv.br/validar-documento. No campo 'Código de registro', informe o código **PERH.593A.CE3V** e clique no botão 'Validar documento'.

Dedico este trabalho aos meus pais, Leda e Carlos, ao meu irmão, Marcel, a minha esposa Vanessa Lopes Moreira e a todos os meus amigos.

AGRADECIMENTOS

Agradeço a Deus por me amparar nos momentos difíceis, me dar força interior para superar as dificuldades e por sempre estar ao meu lado.

À minha mãe, Leda, e ao meu pai, Carlos, por sempre estar ao meu lado com o seu amor eterno e incondicional. Obrigado por terem me educado, pelos conselhos, incentivos, afeto, amizade, dedicação e por não medirem esforços para tornar meus sonhos possíveis. Se hoje cheguei aqui, tenho muito a agradecê-los. Obrigado ao meu irmão, Marcel, pela cumplicidade e amizade.

À minha família, a qual amo muito, pelo carinho, paciência e incentivo.

À Universidade Federal de Viçosa pela oportunidade de realizar o curso. Ao meu orientador, Thiago Henrique Nogueira, por sua dedicação, paciência, confiança, por seus ensinamentos e por me mostrar que eu era capaz. Agradeço também ao Jose Elias Arroyo pela sua dedicação e ensinamentos. Agradeço também a todos os docentes do Departamento de Informática da UFV que de alguma forma contribuíram para o meu crescimento intelectual e profissional. Aos funcionários do departamento, pelas diversas assistências e agradáveis momentos de distração.

À Vanessa Lopes Moreira. Sua ajuda e apoio foram para mim de valor inestimável, e qualquer palavra que eu utilize seria insuficiente para agradecê-la com o devido merecimento. Obrigado por acreditar em mim quando eu achei difícil acreditar em mim mesmo. Tenho certeza que sem você eu não teria chegado até aqui.

Por fim, o meu profundo e sentido agradecimento a todas as pessoas que contribuíram para a concretização desta tese, estimulando-me intelectualmente e emocionalmente.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

"Você não consegue ligar os pontos olhando para frente; você só consegue ligá-los olhando para trás. Então você tem que confiar que os pontos se ligarão algum dia no futuro." Steve Jobs

RESUMO

ARAÚJO, Matheus de Freitas, D.Sc., Universidade Federal de Viçosa, outubro de 2024. **Métodos Híbridos de Otimização Combinatória e Aprendizado por Reforço para Problemas Integrados de Produção e Distribuição**. Orientador: Thiago Henrique Nogueira. Coorientador: Jose Elias Claudio Arroyo.

Os problemas integrados de planejamento da produção e distribuição em operações logísticas modernas representa um desafio complexo e relevante para a indústria. A solução conjunta desses problemas é essencial para maximizar a eficiência operacional e assegurar o atendimento eficaz das demandas dos clientes. Neste trabalho abordamos o problema de programação de tarefas em máquinas paralelas não-relacionadas com tempo de preparação dependente da sequência integrado com o problema de roteamento de veículos. Para resolver este problema propomos uma abordagem abrangente que envolve três métodos principais: Modelos de Programação Linear Inteira Mista, Heurísticas e Meta-Heurísticas. Esses métodos foram desenvolvidos especificamente para lidar com a alta complexidade dos problemas industriais e garantir soluções de qualidade elevada.

Um dos avanços significativos deste estudo é o desenvolvimento de um *Framework* que utiliza aprendizado de máquina para prever, de forma adaptativa, a heurística de busca de vizinhança mais adequada para cada instância do problema. Esse *Framework* dinâmico seleciona, em tempo real, a estratégia de otimização mais eficiente, aprimorando a eficácia dos algoritmos implementados. Além disso, introduzimos a Meta-heurística Híbrida *Proximal Policy Optimization* com *Variable Neighborhood Descent* (PPO-VND), que combina aprendizado por reforço profundo com busca local. O PPO-VND destaca-se pela sua adaptabilidade, ajustando dinamicamente a política de busca local com base nas decisões aprendidas durante o processo de otimização. Os resultados obtidos com o PPO-VND foram comparados com os da meta-heurística *Random Variable Neighborhood Descent* (RVND).

Para avaliar a qualidade das soluções obtidas pelos métodos propostos, foram calculados o Desvio Percentual Relativo (RPD) que quantifica a diferença percentual entre a solução obtida e uma solução de referência e o Melhoria Percentual Relativa (RPI) que quantificar a porcentagem de melhoria alcançada por um algoritmo em comparação com a solução inicial. Os experimentos computacionais confirmam a eficácia do PPO-VND, especialmente em instâncias de maior complexidade, onde o método foi capaz de encontrar soluções de alta qualidade em tempos computacionais aceitáveis. A integração de técnicas de aprendizado de

máquina com métodos de otimização combinatória mostrou-se fundamental para promover maior adaptabilidade e robustez em ambientes operacionais dinâmicos, como aqueles encontrados em operações logísticas.
Em conclusão, esta tese demonstra que a adoção de abordagens híbridas, que combinam aprendizado de máquina e técnicas de otimização combinatória, oferece uma solução promissora para problemas complexos, como o planejamento integrado de produção e roteamento de veículos. As contribuições teóricas e práticas apresentadas fornecem uma base sólida para a aplicação dessas metodologias em cenários industriais reais e abrem caminhos para futuras pesquisas, que podem aprofundar a integração de aprendizado por reforço e heurísticas na resolução de problemas de otimização combinatória.

Palavras-chave: problema integrado de planejamento da produção e distribuição. ; programação de máquinas paralelas.; problema de roteamento de veículos.; heurísticas; meta-heurística híbrida.; aprendizado de máquina.; aprendizado não supervisionado. ; aprendizado por reforço.

ABSTRACT

ARAÚJO, Matheus de Freitas, D.Sc., Universidade Federal de Viçosa, October, 2024. **Hybrid Combinatorial Optimization and Reinforcement Learning Methods for Integrated Production and Distribution Problems**. Adviser: Thiago Henrique Nogueira. Co-adviser: Jose Elias Claudio Arroyo.

Integrated production and distribution planning problems in modern logistics operations pose a complex and significant challenge for the industry. The integrated resolution of these problems is essential for maximizing operational efficiency and meeting customer demands effectively. In this work, we address the problem of scheduling tasks on unrelated parallel machines with sequence-dependent setup times, integrated with the capacitated vehicle routing problem. To tackle this issue, we propose a comprehensive approach based on three main methodologies: Mixed Integer Linear Programming (MILP) models, heuristics, and metaheuristics. These methods were meticulously developed to address the high complexity of industrial problems and ensure the generation of high-quality solutions.

A major contribution of this study is the development of a Framework that employs machine learning to predict, adaptively and in real-time, the most suitable neighborhood search heuristic for each problem instance. This dynamic framework selects the most efficient optimization strategy during execution, significantly enhancing the effectiveness of the implemented algorithms. Additionally, we introduce the Hybrid Proximal Policy Optimization with Variable Neighborhood Descent (PPO-VND) metaheuristic, which combines deep reinforcement learning with local search techniques. PPO-VND stands out for its adaptability, dynamically adjusting the local search policy based on insights learned during the optimization process. The performance of PPO-VND was compared with that of the Random Variable Neighborhood Descent (RVND) metaheuristic.

To assess the quality of the solutions generated by the proposed methods, we calculated the Relative Percentage Deviation (RPD), which quantifies the percentage difference between an obtained solution and a reference solution, and the Relative Percentage Improvement (RPI), which measures the percentage improvement achieved by an algorithm over its initial solution. Computational experiments confirm the effectiveness of PPO-VND, particularly in more complex instances, where it achieved high-quality solutions within acceptable computational times. The integration of machine learning techniques with combinatorial optimization methods proved essential for enhancing adaptability and robustness in dynamic operational environments, such as those encountered in logistics

operations.
In conclusion, this thesis demonstrates that hybrid approaches combining machine learning and combinatorial optimization techniques offer a promising avenue for addressing complex problems, such as integrated production planning and vehicle routing. The theoretical and practical contributions presented provide a solid foundation for applying these methodologies in real-world industrial scenarios. Furthermore, they pave the way for future research, which may deepen the integration of reinforcement learning and heuristics in solving combinatorial optimization problems.

Keywords: integrated production and distribution planning problem.; parallel machine scheduling.; vehicle routing problem.; heuristics; hybrid metaheuristics.; machine learning.; unsupervised machine learning.; reinforcement learning.

Lista de Figuras

Lista de Tabelas

Lista de Abreviações e Siglas

A3C Ator-crítico de Vantagem Assíncrona (do inglês, *Asynchronous Advantage Actor-Critic*)
ANOVA Análise de Variância
B&B *Branch-and-Bound*
CPS Sistemas Ciber-Físicos (do inglês, *Cyber-Physical Systems*)
DNN Rede Neural Profunda (do inglês, *Deep Neural Network*)
DRL Aprendizado por Reforço Profundo (do inglês, *Deep Reinforcement Learning*)
EDD Data de Entrega mais Próxima (do inglês, *Earliest Due Date*)
GA Algoritmo Genético (do inglês, *Genetic Algorithm*)
HFVRP Problema de Roteamento de Veículos de Frota Heterogênea (do inglês, *Heterogeneous Fleet Vehicle Routing Problem*)
IA Inteligência Artificial
ILS Pesquisa Local Iterada (do inglês, *Iterated Local Search*)
IoS Internet dos Serviços (do inglês, *Internet of Services*)
IoT Internet das Coisas (do inglês, *Internet of Things*)
JIT Produção Enxuta (do inglês, *Just in Time*)
LB Limite Inferior (do inglês, *Lower Bound*)
LNS Busca em Larga Vizinhança (do inglê *Large Neighborhood Search*)
MA Algoritmo Memético (do inglês, *Memetic Algorithm*)
MDP Processo de Decisão de Markov (do inglês, *Markov Decision Process*)
MILP Programação Linear Inteira Mista (do inglês, *Mixed Integer Linear Programming*)
ML Aprendizado de Máquina (do inglês, *Machine Learning*)

MLP *Multilayer Perceptron*

MSE Erro Quadrático Médio (do inglês, *Mean Squared Error*)

NEH *Nawaz-Enscore-Ham*

NSH Heurística de Busca de Vizinhança (do inglês, *Neighborhood Search Heuristics*)

PG Gradientes de Política (do inglês, *Policy Gradients*)

PIFH *Push Forward Insertion Heuristic*

PPO Otimização de Política Proximal (do inglês, *Proximal Policy Optimization*)

PPO-VND Otimização de Política Proximal - Descida de Vizinhança Variável (do inglês, *Proximal Policy Optimization - Variable Neighborhood Descent*)

RL Aprendizado por Reforço (do inglês, *Reinforcement Learning*)

RPD Desvio Percentual Relativo (do inglês, *Relative Percentage Deviation*)

RPI Melhoria Percentual Relativa (do inglês, *Relative Percentage Improvement*)

RVND Descida de Vizinhança Variável Aleatória (do inglês, *Randomized Variable Neighborhood Descent*)

TWT Atraso Total Ponderado (do inglês, *Total Weighted Tardiness*)

UB Limite Superior (do inglês, *Upper Bound*)

VND Descida de Vizinhança Variável (do inglês, *Variable Neighborhood Descent*)

Sumário

Capítulo 1

Introdução

No cenário atual de evolução tecnológica, marcado pela ascensão da Indústria 4.0, a integração de sistemas ciber-físicos (*Cyber-Physical Systems - CPS*), Internet das Coisas (*Internet of Things - IoT*) e Internet dos Serviços (*Internet of Services - IoS*) tem transformado profundamente os métodos de produção e distribuição. Esta revolução tecnológica visa a criação de fábricas inteligentes, caracterizadas por uma automação total, na qual pessoas, máquinas e recursos comunicam-se de maneira descentralizada, sem a necessidade de um controle central, aumentando tanto a eficiência quanto a flexibilidade das operações industriais. Esta mudança de paradigma destaca o papel crucial da Inteligência Artificial (IA) na otimização dos processos logísticos. A IA aprimora a alocação de recursos, reduz os custos operacionais e eleva a qualidade dos serviços em sistemas descentralizados, onde falhas ou atrasos na entrega podem resultar em perdas significativas e grandes impactos na cadeia de abastecimento.

Neste contexto, a análise preditiva orientada pela IA e a capacidade de tomada de decisão em tempo real tornam-se indispensáveis para fortalecer a resiliência e garantir a continuidade das operações diante das complexidades da Indústria 4.0. Ao priorizar a qualidade do serviço, as organizações podem enfrentar os desafios impostos por sistemas de produção descentralizados, promovendo inovação e aumentando a satisfação dos clientes. A interação simbiótica entre a IA e a Indústria 4.0 possibilita a criação de ecossistemas logísticos dinâmicos, onde eficiência,

flexibilidade e adaptabilidade convergem para atender às crescentes exigências e demandas dos mercados modernos.

No contexto da Indústria 4.0, o sistema *Just in Time* (JIT, Produção Enxuta) se destaca como uma solução para os desafios de operar sem estoque, onde a conectividade contínua ao longo da cadeia de suprimentos torna-se essencial. O princípio fundamental do JIT é produzir apenas o necessário, no momento necessário e com os recursos adequados. Essa abordagem minimiza o excesso de estoque e reduz os custos de armazenamento. No entanto, sua eficácia depende da interligação, do alinhamento e da prontidão de toda a cadeia de abastecimento para atender às demandas de maneira rápida e eficiente, destacando o papel crucial da IA na otimização das operações logísticas dentro do escopo da Indústria 4.0.

Os princípios da produção JIT, adotados pioneiramente pela Toyota (Ghinato, 1995; Monden, 2011; Sugimori et al., 1977), demonstram os benefícios tangíveis das operações simplificadas em indústrias de produção e montagem. Ao minimizar inventários, desperdícios e tempos ociosos, os sistemas JIT aprimoram simultaneamente a eficiência e a eficácia operacional, alinhando-se aos desafios logísticos contemporâneos mencionados anteriormente. Na era da Indústria 4.0, onde sistemas de produção descentralizados exigem alta interconectividade e agilidade, os princípios JIT assumem uma importância ainda maior, ressaltando a necessidade de cadeias de abastecimento sincronizadas e processos otimizados. Essa relação simbiótica entre as metodologias JIT e o panorama logístico destaca o papel essencial da IA na orquestração de operações contínuas, na promoção da eficiência e na garantia de entregas pontuais de bens e serviços em meio às complexidades crescentes.

Nosso estudo investiga as complexidades dos sistemas de produção descentralizados, destacando a importância crucial da integração entre o planejamento da produção e da distribuição. Diferentemente das metodologias convencionais, que se concentram principalmente na redução de custos e na otimização de recursos, nossa pesquisa enfatiza a melhoria dos níveis de serviço ao cliente. Ao priorizar a minimização do atraso total ponderado na entrega dos produtos, nosso estudo se alinha às crescentes demandas das operações logísticas contemporâneas.

Ao aplicar os princípios do JIT e da otimização orientada por IA, enfrentamos as complexidades da Indústria 4.0, promovendo cadeias de fornecimento

sincronizadas e impulsionando a eficiência operacional. Por meio de uma abordagem holística, que enfatiza a qualidade do serviço e a capacidade de resposta, nossa investigação contribui significativamente para o debate contínuo sobre a otimização das operações logísticas no cenário dinâmico da Indústria 4.0.

A otimização combinatória dos processos, envolvendo a programação da produção e o roteamento de veículos, é essencial para alcançar elevados níveis de eficiência operacional e qualidade no atendimento ao cliente. Este problema combinado é notoriamente NP-difícil, o que significa que encontrar soluções ótimas em tempo polinomial é impraticável para instâncias complexas. Portanto, torna-se necessário o desenvolvimento de heurísticas e meta-heurísticas sofisticadas para obter soluções aproximadas de alta qualidade.

A programação da produção e o roteamento de veículos são tradicionalmente tratados como problemas independentes. No entanto, integrar esses dois problemas permite otimizar o sistema como um todo, resultando em melhorias substanciais no desempenho operacional. Estudos recentes destacam a importância dessa abordagem integrada, evidenciando que a coordenação entre produção e distribuição pode reduzir significativamente os custos operacionais e melhorar a qualidade do serviço.

O problema integrado que abordamos envolve a programação de um conjunto de trabalhos em máquinas paralelas não-relacionadas, seguida da distribuição desses trabalhos aos clientes utilizando uma frota de veículos capacitados. Cada trabalho possui tempos de processamento específicos para cada máquina, prazos de entrega, penalidades por atraso e tamanhos que ocupam espaço nos veículos. A complexidade adicional surge dos tempos de preparação dependentes da sequência, que variam conforme a ordem de processamento dos trabalhos em cada máquina. Após o processamento, os trabalhos são agrupados em lotes e entregues aos clientes, respeitando as capacidades dos veículos e minimizando o atraso total ponderado.

Nosso trabalho propõe diversas heurísticas de busca local em vizinhança e um *Framework* que utiliza aprendizado de máquina para prever a melhor estratégia de busca a ser aplicada a uma instância específica do problema. Este *Framework* considera características das instâncias, como o número de trabalhos e máquinas, tempos médios de processamento e preparação, tempos de viagem e capacidades

dos veículos. Ao prever a melhor heurística, o *Framework* aumenta a eficiência da solução, adaptando-se dinamicamente às particularidades de cada instância.

Além disso, introduzimos uma abordagem inovadora que combina o algoritmo de aprendizado por reforço *Proximal Policy Optimization* (PPO) com o algoritmo *Variable Neighborhood Descent* (VND). Esta meta-heurística híbrida, denominada PPO-VND, utiliza o PPO para selecionar dinamicamente a estratégia de busca local mais eficaz em cada iteração do algoritmo VND. Avaliamos essa abordagem por meio de experimentos computacionais em diversas instâncias do problema, demonstrando que o PPO-VND supera métodos tradicionais, como a meta-heurística *Random Variable Neighborhood Descent* (RVND) e o modelo de Programação Linear Inteira Mista (MILP), tanto em eficiência quanto na qualidade das soluções.

Os avanços propostos nesta tese não apenas enriquecem a literatura existente, mas também oferecem soluções práticas para desafios reais enfrentados pelas indústrias modernas. A capacidade de prever a melhor heurística para cada instância específica, aliada à adaptabilidade dinâmica proporcionada pelo aprendizado por reforço, representa um avanço significativo na otimização combinatória de problemas integrados de produção e distribuição. Além das técnicas já mencionadas, nosso trabalho propõe um conjunto de instâncias de teste baseadas em problemas similares encontrados na literatura, permitindo uma avaliação robusta dos métodos propostos e assegurando que as soluções desenvolvidas sejam aplicáveis a uma ampla gama de cenários práticos.

1.1 Objetivo da Tese

O objetivo principal desta tese é desenvolver métodos exatos, heurísticas e meta-heurísticas para resolver o problema integrado de programação de tarefas em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência com o problema de roteamento de veículos capacitados. Para resolver esse problema, utilizamos modelos matemáticos e algoritmos de busca em vizinhanças. Adicionalmente, propomos um *Framework* que utiliza técnicas de aprendizado de máquina para identificar a heurística mais eficaz para resolver cada instância do problema com base em suas características estatísticas. Entre as meta-heurísticas propostas, destacamos a combinação dos algoritmos PPO com o VND, denomi-

nada PPO-VND. Essa abordagem utiliza aprendizado por reforço profundo para prever e selecionar heurísticas de busca de vizinhança apropriadas em cada iteração do algoritmo. Espera-se que esses algoritmos possam encontrar soluções de alta qualidade em um tempo computacional aceitável para um conjunto diversificado de instâncias, demonstrando que as técnicas com aprendizado de máquina são mais eficientes e eficazes do que as que não utilizam, como a RVND. Embora a pesquisa não tenha um foco específico em um setor industrial, os métodos desenvolvidos podem ser aplicados e adaptados a diversos contextos que envolvam a programação e distribuição da produção, oferecendo soluções práticas e eficazes.

1.1.1 Objetivos Específicos

Os objetivos específicos a seguir são utilizados para alcançar o objetivo geral dessa tese:

- Desenvolver modelos matemáticos exatos para a otimização do problema integrado de programação de tarefas em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência e o problema de roteamento de veículos capacitados. Esses modelos fornecerão uma base teórica robusta e precisa para a resolução do problema, assegurando rigor matemático e confiabilidade nos resultados.

- Implementar heurísticas de busca em vizinhanças aplicáveis ao problema, visando encontrar soluções de alta qualidade em tempo computacional adequado para diferentes instâncias do problema. Essas heurísticas serão avaliadas quanto à eficiência e eficácia em diferentes cenários e configurações do problema.

- Desenvolver um *Framework* baseado em aprendizado de máquina para identificar a heurística mais eficaz para cada instância do problema, considerando suas características estatísticas, como número de máquinas e tarefas, tempos médios de preparação e processamento, tempo médio de viagem e valores médios das demandas e tamanho das tarefas. Este *Framework* deverá adaptar-se dinamicamente às particularidades de cada instância, melhorando a qualidade das soluções obtidas.

- Propor e implementar a meta-heurística híbrida PPO-VND, que combina o algoritmo de aprendizado por reforço profundo PPO com o VND. O objetivo é utilizar o aprendizado por reforço para selecionar de forma inteligente as melhores heurísticas de busca de vizinhança a cada iteração, aprimorando assim a eficácia e eficiência do processo de otimização.

- Comparar a eficácia das técnicas propostas com técnicas tradicionais, como a meta-heurística RVND, por meio de experimentos computacionais em diversas instâncias do problema. Avaliar o desempenho das técnicas com e sem aprendizado de máquina, comparando a qualidade das soluções e o tempo computacional, a fim de quantificar os benefícios do aprendizado de máquina na resolução do problema proposto.

- Para avaliar a qualidade das soluções obtidas pelos métodos propostos serão utilizadas duas métricas: o Desvio Percentual Relativo (RPD) que quantifica a diferença percentual entre a solução obtida e uma solução de referência e o Melhoria Percentual Relativa (RPI) que quantificar a porcentagem de melhoria alcançada por um algoritmo em comparação com a solução inicial.

- Validar os modelos e algoritmos desenvolvidos por meio de testes computacionais rigorosos, garantindo que os resultados sejam aplicáveis a diferentes contextos industriais. A validação incluirá a análise de instâncias variadas, permitindo uma avaliação robusta das técnicas propostas.

- Divulgar os resultados da pesquisa em conferências e revistas científicas de relevância na área de otimização combinatória e inteligência artificial. A divulgação visa não apenas contribuir para o avanço da aplicação de algoritmos de aprendizado de máquina combinados com meta-heurísticas na resolução de problemas complexos, mas também permitir que outros pesquisadores e profissionais se beneficiem dos métodos desenvolvidos nesta tese.

1.2 Estrutura da Tese

A organização desta tese segue a estrutura abaixo:

No Capítulo 2, é apresentado a caracterização do problema e uma revisão bibliográfica sobre o problema integrado de programação de tarefas em máquinas paralelas e roteamento de veículos. Este capítulo também aborda estudos relacionados a técnicas de aprendizado de máquina e meta-heurísticas aplicadas na resolução desses problemas de otimização combinatória.

O Capítulo 3 define detalhadamente o problema desta pesquisa. São descritos os problemas de programação de tarefas em máquinas, o problema de roteamento de veículos e a integração entre eles. Além disso, são apresentados os modelos MILP propostos para resolver o problema estudado.

O Capítulo 4 discute os métodos de otimização combinatória utilizados nesta pesquisa. Este capítulo abrange a representação da solução, algoritmos de decodificação, solução inicial e heurísticas de busca de vizinhança. Além disso, este capitulo apresenta o *Framework* baseado em aprendizado de máquina, bem como as meta-heurísticas RVND e a meta-heurística híbrida PPO-VND.

O Capítulo 5 apresenta os experimentos computacionais realizados para validar os métodos propostos. Este capitulo também apresenta as instâncias do problema e os resultados computacionais obtidos. O capítulo inclui comparações entre os modelos MILP, o *Framework* e as heurísticas de busca de vizinhança, além de experimentos específicos para o treinamento do algoritmo PPO e a calibração do algoritmo RVND. Também são discutidos os resultados das comparações entre modelos MILP, Framework, RVND e PPO-VND, assim como uma visão geral dos resultados obtidos.

Finalmente, o Capítulo 6 apresenta as considerações finais da pesquisa. O capítulo resume as principais contribuições do trabalho, discute as limitações e propõe direções para futuras pesquisas com o objetivo de expandir e aprofundar os métodos e abordagens desenvolvidos.

Capítulo 2

Revisão Bibliográfica

Problemas de planejamento da produção e distribuição são frequentes em várias aplicações que envolvem produtos fabricados sob encomenda ou sensíveis ao tempo. Em muitos casos, há pouco ou nenhum estoque disponível, o que torna o planejamento da produção e distribuição altamente dependente da coordenação eficaz entre as etapas de produção e entrega (Chen, 2010). Tradicionalmente, esses problemas são resolvidos de forma independente e, posteriormente, unificados para determinar a solução final, uma abordagem que, sabe-se, não gera uma solução ótima. Nos últimos anos, muitos estudos têm focado na programação integrada de produção e distribuição, um desafio significativo, especialmente para empresas dos setores de manufatura e logística. Essa integração permite a redução dos níveis de estoque por meio de uma otimização global de todo o sistema de produção e distribuição. Os problemas integrados possuem diversas aplicações práticas, como demonstrado por Ullrich (2013a), Yağmur & Kesen (2024), Mohammadi et al. (2020), Boudia et al. (2008).

Chen (2010) fornece uma extensa revisão bibliográfica sobre problemas de programação integrada de produção e distribuição, sintetizando os resultados existentes e apresentando um sistema de representação unificado para os modelos. Chen também classifica os modelos em diferentes categorias, com base em características comuns, e revisa os resultados obtidos para cada classe de modelos. Além disso, o autor aborda alguns problemas ainda sem solução e propõe novas direções de pesquisa. Neste capítulo, apresentamos trabalhos relacionados ao tema, inici-

ando com problemas estudados de forma separada e, por fim, abordando estudos que tratam desses problemas de maneira integrada.

Os problemas de programação da produção têm sido amplamente estudados ao longo das últimas décadas, com diversos trabalhos focados em máquinas paralelas. O primeiro estudo sobre programação da produção em máquinas paralelas foi realizado por McNaughton (1959), seguido por contribuições significativas de Baker & Merten (1973), Coffman (1976) e Lenstra (1977). Cheng & Sin (1990) classificam este problema como NP-completo e apresentam uma abrangente revisão bibliográfica, detalhando as principais variações.

O problema de programação de tarefas em máquinas paralelas não-relacionadas, com tempos de preparação dependentes da sequência, é uma extensão do problema de programação de tarefas em máquinas paralelas. Diversos autores propuseram soluções para esse problema, cada um com objetivos específicos. Rabadi et al. (2006) e Vallada & Ruiz (2011) apresentam, respectivamente, uma meta-heurística e um algoritmo genético para minimizar o *makespan* (o tempo total necessário para completar todas as tarefas). Por outro lado, Kim et al. (2002) propõem uma heurística para minimizar o atraso total, enquanto Logendran et al. (2007) e Kim et al. (2003) propõem algoritmos que se concentram na minimização do atraso total ponderado, variando na forma de cálculo do atraso com base nas penalidades associadas.

Outro problema clássico presente nos problemas integrados de planejamento e distribuição da produção é o problema de roteamento de veículos (VRP, do inglês *Vehicle Routing Problem*). O VRP, introduzido por Dantzig & Ramser (1959), é amplamente aplicável no cotidiano de diversas empresas. Este problema é uma generalização do Problema do Caixeiro Viajante (TSP, do inglês *Travelling Salesman Problem*) e pertence à classe de problemas NP-difíceis, ou seja, problemas para os quais não existe um algoritmo que os resolva em tempo polinomial para todas as instâncias (Cordeau et al., 2007). O problema clássico de roteamento de veículos visa definir rotas de entrega com menor custo entre um depósito e um conjunto de locais geograficamente dispersos (clientes), respeitando um conjunto de restrições como a capacidade dos veículos e o tempo de entrega (Kumar & Panneerselvam, 2012). Cordeau et al. (2007) discutem os principais tipos de problemas de roteamento de veículos, apresentando modelos matemáticos, algorit-

mos exatos e heurísticas. Dentre essas variantes, destaca-se o VRP com múltiplos depósitos e o VRP com restrições de capacidade.

Várias variantes do VRP foram desenvolvidas a partir do problema clássico, como o problema de roteamento de veículos capacitados (CVRP, do inglês *Capacitated Vehicle Routing Problem*), que considera veículos com capacidade limitada para atender as demandas dos clientes (Laporte et al., 1985). No CVRP, utilizam-se veículos idênticos com uma capacidade máxima que deve ser respeitada, além de restrições de distâncias máximas nas rotas. Diversos estudos propõem algoritmos para resolver o CVRP. Toth & Vigo (2002) apresentam uma revisão dos principais algoritmos baseados em *Branch-and-Bound* para o CVRP. Já Lysgaard et al. (2004) propõem um algoritmo de *Branch-and-Cut* para resolver o problema.

Nas últimas décadas, a comunidade científica desenvolveu uma vasta gama de modelos matemáticos, heurísticas, meta-heurísticas, heurísticas híbridas e algoritmos exatos para resolver uma ampla variedade de problemas de programação de produção e roteamento de veículos. As heurísticas híbridas, que combinam diferentes técnicas de otimização, têm mostrado resultados promissores em problemas complexos. Tais abordagens proporcionaram avanços significativos na eficiência e precisão com que esses problemas são resolvidos, refletindo a constante evolução no campo da otimização combinatória.

2.1 Problemas de Otimização Resolvidos com Técnicas de Inteligência Artificial

Recentemente, com o desenvolvimento e a crescente adoção de tecnologias baseadas em IA, abriram novas perspectivas de pesquisa que integram métodos baseados em IA na resolução de problemas de otimização combinatória. Esses estudos inovadores aprimoram as capacidades existentes e abrem caminhos para soluções mais adaptativas e inteligentes, capazes de lidar com a complexidade e as incertezas dos sistemas modernos de produção e logística. O uso de técnicas de aprendizado de máquina e, em particular, aprendizado profundo, tem mostrado grande potencial para superar as limitações dos métodos tradicionais, oferecendo soluções

que se adaptam dinamicamente às mudanças nas condições operacionais. Além disso, a integração de técnicas de IA com métodos tradicionais, como heurísticas e meta-heurísticas, tem mostrado resultados promissores na resolução de problemas complexos de otimização combinatória. Essas abordagens híbridas combinam a capacidade adaptativa da IA com a eficiência das técnicas clássicas de busca, gerando soluções robustas em cenários dinâmicos.

Zhu et al. (2020) propuseram uma abordagem de Aprendizado por Reforço Profundo (DRL, do inglês, *Deep Reinforcement Learning*) para resolver o problema de programação da produção em um ambiente de produção *Flow Shop Flexível* (FFSP), para minimizar o *makespan* e reduzir o tempo necessário para concluir um conjunto de tarefas em máquinas paralelas idênticas operando em paralelo. Os autores adotaram o algoritmo PPO para avaliar e otimizar instâncias complexas e de diferentes escalas, incluindo dados do mundo real e cenários gerados aleatoriamente. Os resultados mostraram que o PPO forneceu consistentemente soluções satisfatórias dentro de limites aceitáveis de tempo computacional.

Nahhas et al. (2022) aplicaram técnicas de DRL para otimizar a programação da produção em um ambiente *Hybrid Flow Shop* (HFS). As técnicas de DRL, incluindo PPO e o algoritmo *Asynchronous Advantage Actor-Critic* (A3C), mostraram-se eficazes no aprendizado de políticas para resolver problemas de HFS e adaptar as estratégias a variados cenários.

Kayhan & Yildiz (2021) apresentaram uma revisão abrangente das aplicações de Aprendizado por Reforço (RL, do inglês, *Reinforcement Learning*) em problemas de planejamento da produção em máquinas, analisando os algoritmos, ambientes, características das máquinas e tarefas, objetivos e métodos comumente empregados. Eles indicaram que os tópicos mais frequentemente investigados incluem problemas de planejamento da produção em oficinas, escalonamento de máquinas paralelas não-relacionadas e escalonamento em máquina única. Suas principais contribuições incluem uma análise detalhada do uso de RL no escalonamento de máquinas, a categorização dos tipos de problemas, objetivos e restrições recorrentes, além da identificação de lacunas na literatura, como a falta de estudos sobre escalonamento com múltiplas restrições.

Além disso, Nazari et al. (2018) e Peng et al. (2020) exploraram o problema simplificado de roteamento de veículos, no qual um único veículo com capacidade

limitada deve atender vários clientes distribuídos geograficamente, cada um com uma demanda finita. O veículo deve retornar ao depósito sempre que a carga se esgotar, buscando minimizar a distância total percorrida enquanto atende todas as demandas dos clientes. Nazari et al. (2018) propôs uma abordagem baseada em RL para resolver o problema, enquanto Peng et al. (2020) aplicou uma abordagem de DRL. Ambos os algoritmos demonstraram resultados significativos em termos de qualidade das soluções e tempo computacional, quando comparados a outros métodos de otimização combinatória.

2.2 Problemas Integrados de Programação da Produção e Distribuição Resolvidos com Métodos Tradicionais de Otimização Combinatória

Os problemas integrados de programação da produção e distribuição são relativamente recentes, mas já existem diversos estudos relevantes na literatura. Tamannaei & Rasti-Barzoki (2019) abordam um problema integrado de programação da produção e roteamento de veículos, determinando o sequenciamento das tarefas em uma máquina simples e sua distribuição por uma frota de veículos heterogêneos. O objetivo é minimizar o atraso total ponderado das tarefas e os custos relacionados ao transporte, utilizando um modelo matemático, o método *Branch-and-Bound* (B&B) e um Algoritmo Genético (GA, do inglês *Genetic Algorithm*). Felix & Arroyo (2020) propõem duas heurísticas híbridas baseadas no *Iterated Local Search* (ILS) e RVND para resolver o problema, comparando seus resultados com os obtidos pelo GA de Tamannaei. Este trabalho é o que mais se aproxima do problema proposto nesta tese, sendo uma importante referência para o desenvolvimento das abordagens integradas.

Liu et al. (2020) estudaram um problema semelhante ao apresentado anteriormente, mas com uma frota de veículos homogênea, com o objetivo de minimizar a soma dos tempos de entrega das tarefas. Eles propuseram uma heurística VNS e compararam os resultados com um modelo matemático, o Algoritmo Memético

(MA, do inglês *Memetic Algorithm*) desenvolvido por Ngueveu et al. (2010), e o Algoritmo *Large Neighborhood Search* (LNS) proposto por Ribeiro & Laporte (2012).

Ta et al. (2015) estudaram um problema integrado utilizando um sistema de produção *flow shop* e um veículo idealizado com capacidade infinita para realizar as entregas. O objetivo é definir o sequenciamento das tarefas nas máquinas, agrupar as tarefas em lotes e determinar as rotas de entrega, buscando minimizar o atraso total. Eles propuseram um modelo matemático e três heurísticas: um Algoritmo Guloso (*Greedy Algorithm*), um Algoritmo de Busca Tabu (TS, do inglês *Tabu Search*) e uma heurística híbrida que combina os dois algoritmos.

Zou et al. (2018) estudaram um problema integrado utilizando uma única máquina e uma frota de veículos com capacidade limitada, visando sequenciar as tarefas na máquina e definir as rotas de entrega para minimizar o tempo total de entrega. Eles propuseram um Algoritmo Genético Aprimorado para resolver o problema, comparando os resultados com a resolução do modelo matemático e outros algoritmos disponíveis na literatura.

Karaoğlan & Kesen (2017) propuseram um algoritmo *Branch-and-Cut* para resolver o problema integrado de planejamento e distribuição da produção. Neste problema, há um único local de produção e um único produto sensível ao tempo, ou seja, com uma vida útil limitada. Os produtos devem ser entregues aos clientes antes do vencimento de suas vidas úteis, respeitando a capacidade máxima dos veículos. O objetivo é minimizar o tempo necessário para produzir e entregar os produtos aos clientes.

Nagano et al. (2022) investigaram um problema integrado de produção e distribuição em um sistema de produção *flow shop* e roteamento de veículos, buscando sequenciar pedidos para minimizar o tempo total de produção. O estudo destaca a importância de operações simplificadas e da utilização eficiente dos recursos produtivos e logísticos.

Martins et al. (2021) investigaram um problema integrado semelhante, focando em um ambiente de produção híbrido de *flow shop* e otimizando o *makespan* com um modelo de programação linear inteira mista e um algoritmo de descida de vizinhança variável aleatória tendenciosa (BR-VND). Por sua vez, Hou et al. (2022) apresentaram um algoritmo de otimização de *brainstorming* aprimo-

rado para minimizar o avanço e o atraso total ponderado. Eles compararam seus resultados com vários outros algoritmos, incluindo o algoritmo discreto de colônia artificial de abelhas (Duan et al., 2018), o algoritmo ganancioso iterativo (Mao et al., 2021), o algoritmo genético (Zhang et al., 2020), o algoritmo memético (Kurdi, 2020) e o algoritmo de pesquisa de dispersão (Pan et al., 2019).

Wang et al. (2020) estudaram um problema de programação de *flow shop* híbrido de três estágios, incorporando a distribuição de produtos. O estudo envolve um sistema com máquinas paralelas idênticas e tempos de configuração dependentes da sequência, bem como máquinas dedicadas, além de abordar o problema do caixeiro viajante com múltiplas viagens para minimizar o tempo máximo de conclusão da entrega. Eles propuseram um modelo de programação linear inteira mista juntamente com a meta-heurística VNS, um método heurístico construtivo de quatro camadas (C-HVNS) e um método heurístico híbrido (CONSVNS).

Geismar et al. (2008) apresentaram um problema integrado de planejamento da produção e distribuição, no qual o *makespan* representa o tempo total necessário para fabricar e entregar os produtos aos clientes. O problema considera uma planta que opera com a produção constante de um único produto, que possui um prazo de validade. O objetivo é determinar o menor *makespan* que permita atender um conjunto de clientes, observando o prazo de validade dos produtos. Os autores apresentaram limites inferiores para a solução ótima do problema e desenvolveram um algoritmo baseado em duas fases para resolvê-lo.

Ullrich (2013b) apresentou um problema integrado de programação da produção e distribuição para minimizar o atraso total. O problema envolve a alocação de tarefas em máquinas paralelas com tempos de preparação que dependem da sequência e a distribuição dessas tarefas com base no problema de roteamento de veículos com janelas de tempo. O autor propõe um algoritmo genético para resolver o problema integrado.

A Tabela 2.1 apresenta uma visão geral dos artigos semelhantes ao nosso trabalho. A tabela inclui colunas que especificam a configuração das máquinas, o tipo de veículos (homogêneo ou heterogêneo, ou seja, veículos iguais ou diferentes) e a quantidade de veículos (veículo único ou frota limitados de veículos). Além disso, a tabela apresenta os nomes dos autores e indicadores que mostram os métodos

utilizados para resolução do problema abordado, esses métodos incluem modelos matemáticos, heurísticas, meta-heurísticas e inteligência artificial.

Com base em estudos de diversos autores que exploraram o planejamento integrado da produção e distribuição, nosso estudo investiga os desafios impostos por sistemas de produção descentralizados, como a coordenação de recursos e a otimização logística. Além desses esforços, nosso estudo enfatiza a otimização dos níveis de serviço ao cliente, integrando metodologias JIT e técnicas de otimização baseadas em IA para lidar com as complexidades da produção descentralizada e da distribuição.

Tabela 2.1. Visão Geral de Artigos sobre Planejamento Integrado de Produção e Distribuição

Configuração das Máquinas	Problema de Roteamento de Veículos		Autor(es)	Modelo Matemático	Heurística	Meta-heurística	IA
	Tipo	Número					
Flexible flowshop	Apenas problema de agendamento		Zhu et al. (2020)	Sim	Sim	Sim	Sim
Hybrid Flow Shop	Apenas problema de agendamento		Nahhas et al. (2022)	Não	Não	Não	Sim
Apenas problema de roteamento de veículos	Homogêneo	Único	Peng et al. (2020) Nazari et al. (2018)	Não	Não	Não	Sim
Flow shop	Homogêneo	Limitado	Hou et al. (2022)	Sim	Não	Sim	Não
Flow shop	Homogêneo	Único	Ta et al. (2015)	Sim	Sim	Não	Não
Permutation flow-shop	Homogêneo	Único	Nagano et al. (2022)	Sim	Não	Sim	Não
Hybrid flow-shop	Homogêneo	Único	Martins et al. (2021), Wang et al. (2020)	Sim	Não	Sim	Não
Flexible job-shop	Heterogêneo	Limitado	Mohammadi et al. (2020)	Sim	Não	Sim	Não
Single machine	Homogêneo	Limitado	Tamannaei & Rasti-Barzoki (2019), Liu et al. (2020)	Sim	Sim	Sim	Não
Single machine	Heterogêneo	Limitado	Felix & Arroyo (2020), Félix et al. (2023), Zou et al. (2018)	Sim	Sim	Sim	Não
Identical parrallel machines	Heterogêneo	Limitado	Yağmur & Kesen (2024)	Sim	Não	Sim	Não
Parallel machines with machine-dependent ready times	Heterogêneo	Limitado	Ullrich (2013a)	Sim	Não	Sim	Não
Unrelated parallel machines with machine-dependent ready times	Heterogêneo	Limitado	Nossa Abordagem	Sim	Sim	Sim	Sim

Fonte: Próprio Autor

Capítulo 3

Definição do Problema

Neste capítulo, abordamos a definição dos principais problemas deste estudo, enfatizando suas características e complexidades. Compreender esses problemas em detalhes é fundamental para desenvolver soluções eficazes e aplicá-las de forma prática em cenários reais. Inicialmente, apresentamos o problema de programação de tarefas, que é a base de muitos desafios de otimização em ambientes industriais. Em seguida, apresentamos o problema de programação de tarefas em máquinas paralelas, destacando suas especificidades e desafios adicionais.

Também discutimos o VRP, um clássico problema de otimização de ampla aplicabilidade em operações logísticas e de distribuição. O VRP é fundamental para definir as rotas mais eficientes para uma frota de veículos, minimizando os custos operacionais e os tempos de viagem.

Em seguida, exploramos os problemas integrados de planejamento da produção e roteamento de veículos. Essa abordagem visa otimizar simultaneamente a produção e a distribuição, superando as limitações das abordagens individuais e proporcionando uma visão holística dos processos produtivos e logísticos. A integração desses problemas é essencial para alcançar maior eficiência operacional e reduzir os custos totais.

Por fim, apresentamos os modelos MILP, utilizados para formalizar e resolver os problemas abordados. Os modelos MILP são ferramentas eficazes para a otimização combinatória, permitindo a modelagem precisa de restrições e objetivos complexos e amplamente utilizados para encontrar soluções ótimas.

Este capítulo estabelece a base teórica fundamental para a compreensão das técnicas de solução propostas nos capítulos subsequentes, fornecendo uma visão abrangente dos desafios e métodos de otimização explorados nesta pesquisa.

3.1 Problema de Programação de Tarefas

Os desafios relacionados ao problema de sequenciamento de tarefas em máquinas têm sido um foco significativo de pesquisa nas últimas seis décadas. A atenção contínua de pesquisadores e profissionais, e o alto volume de soluções desenvolvidas, refletem a importância e a complexidade desses problemas. O sequenciamento de tarefas em máquinas impacta diretamente a qualidade do serviço e os aspectos econômicos, sendo crucial para as indústrias em um mercado altamente competitivo.

Nos problemas de sequenciamento de tarefas, é essencial especificar tanto as restrições tecnológicas associadas às tarefas quanto os objetivos de sequenciamento. Pinedo (2016) utiliza as restrições tecnológicas, que definem o fluxo de tarefas entre as máquinas, para classificar os problemas da seguinte maneira:

- Máquina Única (*Single Machine Scheduling*): O problema de programação da produção em máquina única é fundamental na área de otimização combinatória e sequenciamento de tarefas. Nesse cenário, todas as tarefas precisam ser processadas em uma única máquina, e o objetivo é determinar a melhor sequência de execução das tarefas para otimizar um critério de desempenho.

- Máquinas Paralelas (*Parallel Machine Scheduling*): Neste caso, o problema envolve alocar e sequenciar as tarefas em várias máquinas que operam em paralelo, para otimizar um critério de desempenho. Nesse contexto, o problema pode envolver máquinas idênticas, uniformes ou heterogêneas.

- *Job Shop Scheduling*: Nos problemas de *Job Shop Scheduling*, cada tarefa segue uma ordem de processamento específica nas máquinas. O problema consiste em organizar um conjunto de tarefas de forma que cada uma seja executada nas máquinas, respeitando a sequência definida de operações. As-

sim como nos demais problemas, cada máquina processa apenas uma tarefa por vez. O objetivo é otimizar um critério de desempenho específico.

- *Flow Shop Scheduling*: O problema de *flow shop scheduling* envolve a organização de um conjunto de tarefas, em que todas seguem a mesma sequência de operações em várias máquinas. Cada máquina processa apenas uma tarefa por vez. As operações ocorrem em uma ordem específica, e todas as tarefas passam pelas máquinas na mesma sequência. O objetivo é otimizar um critério de desempenho específico.

- *Open Shop Scheduling*: O problema de *open shop scheduling* envolve a organização de um conjunto de tarefas, em que cada tarefa é processada em várias máquinas, mas sem uma ordem fixa de operações. As tarefas podem ser executadas em qualquer ordem nas máquinas, e cada máquina processa apenas uma tarefa por vez. O objetivo é otimizar um critério de desempenho específico.

Diversas variações dos problemas de sequenciamento de tarefas mencionados podem ser apresentadas, como o *flow shop scheduling* permutacional, o *job shop scheduling* com múltiplas máquinas, o *flow shop scheduling* com múltiplas máquinas, o *flexible job shop scheduling*, o *batch scheduling* e o *hybrid flow shop scheduling*. Entre esses, o problema de programação da produção em máquina única é o caso mais simples, uma vez que envolve a ordenação de n tarefas, resultando em $n!$ (fatorial de n) soluções possíveis.

Embora alguns problemas menores possam ser resolvidos de maneira simples utilizando programação linear inteira para alcançar a solução ótima, a maioria dos problemas de sequenciamento de produção é intrinsecamente complexa. Grande parte desses problemas é classificada como NP-difícil, o que significa que não podem ser resolvidos em tempo polinomial por algoritmos conhecidos. Lenstra et al. (1977) apresentam a complexidade de diversos problemas de sequenciamento, incluindo o caso mais simples de programação da produção em máquina única para minimizar o atraso total, classificado como NP-difícil. Os problemas de sequenciamento de tarefas podem ter diferentes objetivos, sendo os principais: minimização do *makespan*, minimização da soma dos tempos de conclusão (*completion time*),

minimização do atraso total e minimização do atraso máximo das tarefas (Pinedo, 2016).

3.2 Problema de Programação de Tarefas em Máquinas Paralelas

O problema de programação de tarefas em máquinas paralelas (*Parallel Machine Scheduling Problem*) é uma extensão do problema clássico de sequenciamento de tarefas, no qual as tarefas devem ser alocadas e sequenciadas em múltiplas máquinas que operam simultaneamente. Este problema surge frequentemente em contextos industriais e de manufatura, onde várias máquinas estão disponíveis para processar tarefas semelhantes, exigindo o balanceamento das máquinas para aumentar a eficiência e produtividade.

O problema de programação de tarefas em máquinas paralelas foi inicialmente introduzido na literatura científica na década de 1960, em resposta à crescente complexidade dos sistemas de produção e à necessidade de métodos mais sofisticados de gerenciamento de tarefas. Uma das primeiras referências notáveis é o trabalho de Conway, Maxwell e Miller no livro *Theory of Scheduling* (Conway, 1967), no qual exploram várias configurações de máquinas paralelas e suas implicações na programação de tarefas.

O problema de programação de tarefas em máquinas paralelas pode ser formalmente definido da seguinte maneira: dado um conjunto $I = \{1, 2, \ldots, n\}$ de n tarefas e um conjunto $M = \{1, 2, \ldots, m\}$ de m máquinas paralelas, o objetivo é sequenciar as tarefas nas máquinas a fim de otimizar um critério de desempenho específico. Neste contexto, as máquinas podem ser idênticas, uniformes ou heterogêneas, dependendo das capacidades e características de processamento.

- Máquinas Idênticas: Todas as máquinas possuem a mesma capacidade e velocidade de processamento, de forma que as tarefas têm o mesmo tempo de processamento em qualquer máquina.

- Máquinas Uniformes: As máquinas têm diferentes capacidades, e o tempo de processamento das tarefas varia proporcionalmente à capacidade de cada máquina.
- Máquinas Não-Relacionadas: Cada máquina possui tempos de processamento diferentes para as tarefas.

Os objetivos do problema podem incluir a minimização do tempo total de conclusão (*makespan*), a minimização do atraso total, ou a maximização da utilização das máquinas. A formulação matemática do problema varia conforme as restrições e objetivos específicos. Em muitos casos, o problema é NP-difícil, exigindo o uso de algoritmos heurísticos e meta-heurísticos para encontrar soluções práticas e eficientes.

Diversos estudos ao longo das últimas décadas abordaram o problema de programação de tarefas em máquinas paralelas. Por exemplo, Allahverdi et al. (2008) e Leung (2004) realizam uma revisão abrangente das técnicas de otimização aplicadas a problemas de programação de tarefas, com foco em máquinas paralelas. Uma contribuição importante é o estudo de Michael (1995), que explora métodos de resolução e otimização em *scheduling*, destacando suas aplicações práticas em ambientes industriais.

3.2.1 Conceitos de tempo de preparação (*setup time*)

O conceito de tempo de preparação (*setup time*) surge da necessidade de preparar uma máquina para processar uma tarefa durante o sequenciamento, dando origem ao problema de programação de tarefas em máquinas paralelas com tempo de preparação. Essa variação do problema é crucial em contextos industriais, onde o tempo de preparação da máquina, como troca de ferramentas, ajustes de configuração ou limpeza entre tarefas, influencia diretamente a eficiência e produtividade do sistema. Allahverdi et al. (1999) realizaram uma revisão abrangente da pesquisa sobre programação de tarefas em máquinas com considerações de tempo de preparação, destacando a relevância desse fator em ambientes de produção, especialmente onde a diversidade de produtos e a frequência de trocas aumentam sua importância.

Em sistemas de programação de tarefas com máquinas paralelas, o objetivo é alocar e sequenciar as tarefas nas máquinas disponíveis, minimizando um critério de desempenho, como o tempo de processamento. Geralmente, o sequenciamento visa equilibrar a carga de trabalho entre as máquinas. A inclusão do tempo de preparação modifica significativamente a abordagem de programação de tarefas, pois, além de equilibrar a carga de trabalho e minimizar o tempo total de processamento, é necessário otimizar a sequência das tarefas para reduzir os tempos de preparação entre elas. Isso aumenta a complexidade do problema, tornando um problema já NP-difícil ainda mais desafiador.

Diversos estudos têm abordado esse tema ao longo das últimas décadas. Por exemplo, Pinedo (2016) oferece uma visão abrangente de teorias, algoritmos e sistemas de *scheduling*, incluindo problemas de máquinas paralelas e tempos de preparação. O trabalho de Cheng et al. (2000) apresenta uma revisão detalhada dos problemas de alocação de tarefas em máquinas paralelas, com ênfase nas técnicas de otimização e heurísticas aplicáveis.

Neste trabalho, estuda-se parcialmente o problema de programação de tarefas em máquinas paralelas não-relacionadas com tempo de preparação dependente da sequência. Nesse contexto, o tempo de preparação (*setup time*) varia dependendo da sequência das tarefas e da máquina utilizada.

Considere um conjunto de tarefas $I = \{1, 2, \ldots, n\}$ a ser processado, sem preempção, em uma das máquinas do conjunto $M = \{1, 2, \ldots, m\}$ de máquinas paralelas não-relacionadas, com tempo de preparação dependente da sequência. Cada tarefa j requer um tempo de processamento p_{ij} na máquina i. As tarefas devem ser processadas sem preempção nas máquinas, ou seja, uma vez alocada na máquina, a tarefa deve ser finalizada sem interrupções. As máquinas são consideradas não-relacionadas quando o tempo de processamento das tarefas depende da máquina à qual estão atribuídas. Este é o caso mais realista, sendo uma generalização dos casos de máquinas uniformes e idênticas.

Os tempos de preparação considerados dependem tanto da sequência das tarefas quanto da máquina em que são processadas. Especificamente, o tempo de preparação na máquina M_i entre as tarefas I_j e I_k, denotado por s_{ijk}, é diferente do tempo de preparação na ordem inversa, ou seja, $s_{ijk} \neq s_{ikj}$. Além disso, o tempo de preparação s_{ijk} na máquina M_i é diferente de $s_{i'jk}$ na máquina $M_{i'}$. Além disso,

a consideração dos tempos de preparação entre as tarefas é uma prática comum na indústria.

O modelo matemático de programação linear inteira mista para o problema de programação de tarefas em máquinas paralelas não-relacionadas, com tempos de preparação dependentes da sequência e o objetivo de minimizar o *makespan*, foi proposto por Avalos-Rosales et al. (2015) e é apresentado a seguir. A tabela 3.1 apresenta os parâmetros e as variáveis de decisão:

Tabela 3.1. Parâmetros e variáveis de decisão do modelo para o problema de programação de tarefas em máquinas paralelas não-relacionadas com tempo de preparação dependente da sequência.

Índices	
i	Máquinas.
j,k	Tarefas.
Conjuntos	
$M = \{1,2,\ldots,m\}$	Conjunto contendo m máquinas.
$I = \{1,2,\ldots,n\}$	Conjunto contendo n tarefas.
$N = I \cup \{O\}$	Conjunto contendo tarefas e a uma tarefa fictícia da origem.
Parâmetros	
p_{ij}	Tempo de processamento da tarefa j na máquina i.
s_{ijk}	Tempo de preparação para a tarefa k após a tarefa j na máquina i.
V	Constante suficientemente grande.
Variáveis de decisão	
X_{ijk}	1 - Se a tarefa j precede a tarefa k na máquina i, 0 - Caso contrário.
Y_{ij}	1 - Se a tarefa j é processada na máquina i, 0 - Caso contrário.
C_j	Tempo de conclusão da tarefa j.

Fonte: Próprio Autor

A seguir, é apresentada a formulação do modelo matemático. O objetivo do modelo é minimizar o *makespan*, definido pela Equação 3.1:

$$\min C_{max} \tag{3.1}$$

Sujeito a:

$$\sum_{j \in N_0, k \in N, k \neq j} s_{ijk} X_{ijk} + \sum_{j \in N} p_{ij} Y_{ij} \leq C_{\max}, \quad i \in M \tag{3.2}$$

$$\sum_{k \in N} X_{i0k} \leq 1, \quad i \in M \tag{3.3}$$

$$\sum_{i \in M} Y_{ij} = 1, \quad j \in N \tag{3.4}$$

$$Y_{ij} = \sum_{k \in N_0, j \neq k} X_{ijk}, \quad i \in M, j \in N \tag{3.5}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk}, \quad i \in M, k \in N \tag{3.6}$$

$$C_k - C_j + V(1 - X_{ijk}) \geq s_{ijk} + p_{ik}, \quad j \in N_0, k \in N, j \neq k, i \in M \tag{3.7}$$

$$C_0 = 0 \tag{3.8}$$

$$C_{\max} \geq C_j, \quad j \in N \tag{3.9}$$

$$X_{ijk} \in \{0, 1\} \quad \forall j \in N, \forall k \in N, j \neq k, \quad \forall i \in M \tag{3.10}$$

$$Y_{ij} \geq 0, \quad \forall j \in I, \quad \forall i \in M \tag{3.11}$$

$$C_j \geq 0 \quad \forall j \in I \tag{3.12}$$

As restrições (3.2) definem o *makespan*. As restrições (3.3) garantem que no máximo uma tarefa seja processada como a primeira em cada máquina, após a tarefa fictícia. As restrições (3.4) asseguram que cada tarefa seja processada em exatamente uma máquina. As restrições (3.5) garantem que todas as tarefas tenham uma sucessora (possivelmente a tarefa fictícia, pois este modelo inclui tarefas fictícias ao final de cada máquina) na máquina em que são processadas. As restrições (3.6) garantem que todas as tarefas tenham uma predecessora na

máquina em que são processadas. As restrições (3.7) fornecem uma ordem de processamento correta. Elas impõem que, se a tarefa k é a sucessora da tarefa j na máquina i, então a tarefa k deve ser concluída em pelo menos $s_{ijk} + p_{ik}$ unidades de tempo após a tarefa j ser concluída. A restrição (3.8) define o tempo de conclusão do trabalho fictício como zero. As restrições (3.9) são cortes viáveis que foram demonstradas como eficientes em (Avalos-Rosales et al., 2015). As restrições 3.10, 3.11 e 3.12 definem a natureza das variáveis.

3.3 Problema de Roteamento de Veículos

O VRP é um dos desafios mais importantes e complexos na área de otimização combinatória. Esse problema foi introduzido por Dantzig & Ramser (1959) e envolve a determinação das rotas mais eficientes para uma frota de veículos que realiza entregas ou coletas em diferentes pontos, considerando restrições como a capacidade dos veículos e os horários de atendimento aos clientes. O objetivo principal é minimizar o custo total das operações logísticas, que pode incluir distâncias percorridas, tempo de viagem ou outras métricas relevantes. Desde sua introdução, o VRP evoluiu significativamente, abrangendo várias variantes que ampliam sua complexidade e aplicabilidade.

A complexidade do VRP é destacada por sua classificação como um problema NP-difícil. Isso significa que, à medida que o número de veículos, clientes e restrições aumenta, o número de possíveis soluções cresce exponencialmente, tornando impraticável obter a solução ótima em tempo razoável para instâncias grandes. Essa complexidade exige o desenvolvimento de métodos de otimização avançados. Métodos exatos, como modelos MILP, podem encontrar soluções ótimas, mas são inviáveis para instâncias de grande escala devido ao elevado tempo computacional necessário para resolver o modelo. Por isso, heurísticas e meta-heurísticas são essenciais para solucionar o VRP de forma eficiente (Lenstra & Kan, 1981).

As aplicações do VRP são vastas e abrangem diversos setores da economia. Na distribuição de produtos de consumo, o VRP é crucial para otimizar a entrega dos centros de distribuição aos pontos de venda, garantindo maior eficiência e economia. Empresas de logística dependem de algoritmos de roteamento para planejar eficientemente suas operações diárias, otimizando rotas de entrega para

economizar combustível, reduzir o tempo de viagem e aumentar a satisfação dos clientes. Na coleta de resíduos urbanos, o VRP é utilizado para otimizar as rotas dos caminhões de lixo, gerando economias significativas para as cidades. No setor de saúde, a distribuição eficiente de medicamentos e equipamentos médicos é essencial para garantir a entrega de produtos sensíveis nos prazos estipulados e nas condições adequadas (Golden et al., 2008).

Neste trabalho, estudamos o CVRP. O objetivo do CVRP é determinar a sequência de visitas aos clientes para cada veículo, de modo que as demandas dos clientes sejam atendidas e o tempo total de transporte seja minimizado, respeitando todas as restrições operacionais. Consideramos um conjunto $L = \{1, 2, \ldots, l\}$ de l veículos utilizados para realizar as entregas, onde cada veículo $v \in L$ possui uma capacidade (q_v) que deve ser respeitada. Temos também um conjunto de clientes $I = \{1, 2, \ldots, n\}$, com cada cliente j possuindo uma demanda a ser atendida (h_j).

Para modelar o problema, definimos um grafo $G(V, A)$, onde $V = \{0, 1, \ldots, n\}$ é o conjunto de nós e $A = \{(j, k) : 0 \leq j, k \leq n, j \neq k\}$ representa o conjunto de arcos. O nó 0 representa o depósito, e os outros nós, os clientes. Cada arco (j, k) possui um tempo de viagem t_{jk} associado, que indica o tempo gasto pelo veículo para percorrer a distância entre os clientes j e k. O objetivo do CVRP é definir as rotas dos veículos para minimizar o tempo total de viagem. Assumimos que cada veículo seja utilizado uma vez, que cada cliente seja atendido por um único veículo, e que as rotas dos veículos comecem e terminem no depósito.

O modelo MILP para o CVRP, visando minimizar a soma dos tempos de viagem, foi proposto por Laporte & Nobert (1987) e é apresentado a seguir. A Tabela 3.2 apresenta os parâmetros e as variáveis de decisão do modelo.

Tabela 3.2. Parâmetros e variáveis de decisão do modelo para o CVRP.

Índices	
v	Veículo.
j,k	Clientes.
Conjuntos	
$L = \{1, 2, ..., l\}$	Conjunto contendo l veículos.
$I = \{1,2,\ldots,n\}$	Conjunto contendo n clientes.
$N = I \cup \{O\}$	Conjunto contendo o deposito e os clientes.
Parâmetros	
h_j	Demanda do cliente j.
q_v	Capacidade do veículo v.
t_{jk}	Tempo de Viagem entre os clientes j e k.
Variáveis de decisão	
Z_{vjk}	1 - Caso o veículo v é utilizado para percorrer entre os pontos j e k 0 - Caso contrário.
U_v	1 - Caso o veículo v é utilizado para realizar as entregas. 0 - Caso contrário.

Fonte: Próprio Autor

A seguir, é apresentada a formulação do modelo matemático. O objetivo do modelo é minimizar o tempo total de transporte, conforme definido pela Equação 3.13:

$$\min \sum_{j,k \in N} \sum_{v \in L} c_{vjk} Z_{vjk} \tag{3.13}$$

sujeito a:

$$\sum_{v \in L} \sum_{k \in N, k \neq j} Z_{vjk} = 1, \forall j \in I \tag{3.14}$$

$$\sum_{j \in N} \sum_{k \in N, k \neq j} Z_{vjk} h_j \leq q_v U_v, \forall v \in L \tag{3.15}$$

$$h_0 = 0 \tag{3.16}$$

$$\sum_{k \in N} Z_{v0k} = U_v, \forall v \in L \tag{3.17}$$

$$\sum_{j \in N} Z_{vjh} = \sum_{k \in N} Z_{vhk} \forall h \in N, \forall v \in L \tag{3.18}$$

$$Z_{vjk} \in 0, 1, \quad \forall j \in N, k \in N, j \neq k, \quad v \in L \tag{3.19}$$

$$U_v \in \{0, 1\} \quad \forall v \in L \tag{3.20}$$

A função objetivo, definida pela Equação 3.13, minimiza os custos relacionados às rotas dos veículos. As restrições 3.14 garantem que cada cliente seja visitado uma única vez. As restrições 3.15 estão relacionadas à ocupação dos veículos e garantem que a capacidade máxima não seja ultrapassada. As restrições 3.16 garantem que o depósito N_0 não ocupe espaço no veículo. As restrições 3.17 garantem que, se um veículo for utilizado, ele deve iniciar a rota a partir do depósito. As restrições 3.18 garantem que um veículo sempre termine a rota no depósito. As restrições 3.19 e 3.20 definem a natureza das variáveis.

3.4 Problema Integrado de Planejamento da Produção e Roteamento de Veículos

No presente trabalho, propomos um problema integrado de planejamento da produção em máquinas paralelas não-relacionadas, com tempos de preparação dependentes da sequência, combinado com o problema de roteamento de veículos capacitados, visando minimizar o atraso total ponderado. Essa combinação é complexa e relevante em contextos industriais onde a eficiência na produção e distribuição de produtos se torna crucial. O problema consiste em determinar a programação de um conjunto de tarefas em máquinas paralelas e, posteriormente, a distribuição das tarefas acabadas aos clientes pelos veículos. A integração dessas duas etapas acrescenta uma camada de complexidade e intensifica a necessidade de soluções otimizadas, devido às interdependências entre produção e distribuição. Este problema é NP-difícil, pois é uma generalização tanto do problema de planejamento da produção em máquinas paralelas não-relacionadas, com tempos de preparação dependentes da sequência, quanto do problema de roteamento de veículos, como

apresentado anteriormente. Por isso, a combinação desses dois subproblemas herda essa complexidade.

Neste problema existe um conjunto de tarefas $I = \{1, 2, \ldots, n\}$ a ser processado, sem preempção, em uma das máquinas do conjunto $M = \{1, 2, \ldots, m\}$ de máquinas paralelas não-relacionadas, com tempos de preparação dependentes da sequência. Cada tarefa j requer um determinado tempo de processamento p_{ij} na máquina i. Além disso, cada tarefa possui um tempo de preparação s_{ijk} que dependem da sequência das tarefas (j e k) e da máquina (i).

Cada tarefa j possui uma data de entrega (d_j), cujo não cumprimento acarreta uma penalidade (w_j), e uma demanda (h_j), que representa o espaço ocupado pela tarefa em um veículo. Após o processamento nas máquinas, as tarefas são divididas em lotes. Cada lote é entregue aos respectivos clientes usando um conjunto $L = \{1, 2, \ldots, l\}$ de l veículos. Para o problema de roteamento de veículos, define-se o grafo $G(V, A)$, onde $V = \{0, 1, \ldots, n\}$ é o conjunto de nós e $A = \{(j,k) : 0 \leq j,k \leq n, j \neq k\}$ é o conjunto de arcos. O nó 0 representa o depósito, e cada tarefa está associada a um cliente, formando o conjunto de clientes $I = \{1, 2, \ldots, n\}$. Além disso, um tempo de viagem t_{jk} está associado a cada arco (j, k). Cada veículo $v \in L$ possui uma capacidade (q_v), que não pode ser excedida. O problema pressupõe que cada veículo seja utilizado uma vez e que cada cliente seja visitado apenas uma vez. O objetivo é determinar as ordens de processamento das tarefas e as rotas de entrega dos veículos para minimizar o atraso total ponderado (TWT, do inglês *Total Weighted Tardiness*) das tarefas.

Para exemplificar melhor o problema, apresenta-se uma instância com três máquinas ($m = 3$), seis tarefas ($n = 6$) e quatro veículos ($v = 4$). As Tabelas 3.3, 3.4 e 3.5 apresentam os parâmetros deste problema. A Figura 3.1 ilustra uma solução para o problema, mostrando a ordem de execução das tarefas, as rotas de entrega, os tempos de conclusão nas máquinas e as datas de entrega aos respectivos clientes. Neste caso, os tempos de conclusão e as datas de entrega são: para a tarefa I_1, 21 e 44; para a tarefa I_2, 45 e 62; para a tarefa I_3, 39 e 59; para a tarefa I_4, 14 e 36; para a tarefa I_5, 8 e 58; e, finalmente, para a tarefa I_6, 23 e 61.

O veículo V_1 inicia a viagem no tempo 14 e é responsável pela entrega das tarefas I_4 e I_5, seguindo a rota: $I_0 \rightarrow I_4 \rightarrow I_5 \rightarrow I_0$, onde I_0 representa o depósito. O veículo V_2 inicia sua rota no tempo 25 e é responsável pela entrega das tarefas I_1

e I_6, seguindo a rota: $I_0 \rightarrow I_1 \rightarrow I_6 \rightarrow I_0$. O veículo V_3 inicia a rota no tempo 45 e é responsável pela entrega da tarefa I_2, seguindo a rota: $I_0 \rightarrow I_2 \rightarrow I_0$. O veículo V_4 inicia a rota no tempo 39 e entrega a tarefa I_3, seguindo a rota: $I_0 \rightarrow I_3 \rightarrow I_0$. Neste exemplo, somente a tarefa I_3 sofreu um atraso de quatro unidades de tempo; a penalidade para cada unidade de tempo é de três unidades. Portanto, o atraso total ponderado (TWT) é de doze unidades (TWT = 12).

Tabela 3.3. Parâmetros das tarefas para o exemplo envolvendo três máquinas, seis tarefas e quatro veículos.

Tarefas	Tempo de processamento			Informação das Tarefas			Distâncias (t_{jk})							
	M_1	M_2	M_3	h_j	d_j	w_j	Tarefas	I_0	I_1	I_2	I_3	I_4	I_5	I_6
I_0	0	0	0	0	0	0	I_0	0	19	17	20	22	22	16
I_1	36	1	43	24	47	10	I_1	19	0	18	34	41	36	17
I_2	46	20	32	27	62	9	I_2	17	18	0	18	33	39	29
I_3	39	24	37	24	55	3	I_3	20	34	18	0	21	38	36
I_4	41	14	32	20	62	2	I_4	22	41	33	21	0	22	33
I_5	7	5	8	5	62	9	I_5	22	36	39	38	22	0	19
I_6	31	45	9	19	61	16	I_6	16	17	29	36	33	19	0

Fonte: Próprio Autor

Tabela 3.4. Tempos de preparação das tarefas para o exemplo envolvendo três máquinas, seis tarefas e quatro veículos.

	Tempo de Preparação M_1								Tempo de Preparação M_2								Tempo de Preparação M_3							
Tarefas	I_0	I_1	I_2	I_3	I_4	I_5	I_6	Tarefas	I_0	I_1	I_2	I_3	I_4	I_5	I_6	Tarefas	I_0	I_1	I_2	I_3	I_4	I_5	I_6
I_0	0	0	0	0	0	0	0	I_0	0	0	0	0	0	0	0	I_0	0	0	0	0	0	0	0
I_1	0	0	1	4	5	2	4	I_1	0	0	4	1	4	5	2	I_1	0	0	4	6	2	3	6
I_2	0	5	0	3	6	4	6	I_2	0	4	0	3	4	1	4	I_2	0	4	0	8	4	3	2
I_3	0	2	3	0	3	1	3	I_3	0	6	3	0	7	4	4	I_3	0	7	6	0	6	7	3
I_4	0	4	2	3	0	4	1	I_4	0	6	2	5	0	3	6	I_4	0	2	3	4	0	1	5
I_5	0	1	2	4	6	0	2	I_5	0	3	1	4	3	0	5	I_5	0	1	4	5	1	0	6
I_6	0	3	1	2	5	3	0	I_6	0	2	4	3	5	3	0	I_6	0	5	3	6	3	4	0

Fonte: Próprio Autor

Tabela 3.5. Informações sobre os veículos para o exemplo envolvendo três máquinas, seis tarefas e quatro veículos.

Informações dos veículos				
Veículo	V_1	V_2	V_3	V_4
q_v	58	65	41	72

Fonte: Próprio Autor

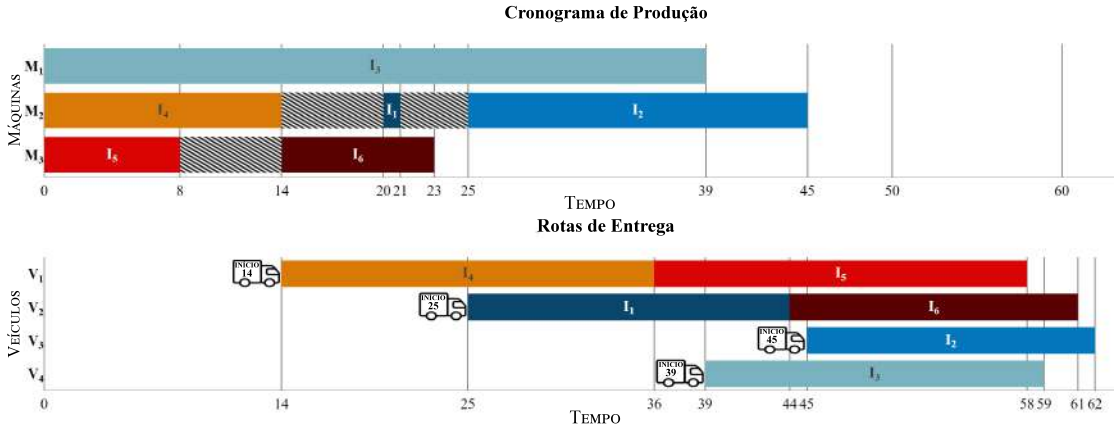


Figura 3.1. O cronograma de produção das tarefas nas máquinas e rotas de entrega (solução ótima) para o exemplo envolvendo três máquinas, seis tarefas e quatro veículos.

Fonte: Próprio Autor

3.5 Modelos Matemáticos de Programação Linear Inteira Mista Propostos

Esta seção apresenta as formulações matemáticas desenvolvidas para abordar o problema estudado de forma integrada. Esses modelos têm como objetivo capturar as complexidades da interação entre a produção e a distribuição, considerando as restrições do problema apresentadas anteriormente. Através da utilização de Programação Linear Inteira Mista (MILP), buscamos fornecer soluções exatas que garantam a otimização global do sistema, levando em conta tanto os aspectos operacionais da produção quanto as exigências logísticas de distribuição. Para esse problema são propostos dois modelos MILP para resolver o problema, o Modelo MILP 1 adota uma abordagem de precedência, utilizando variáveis binarias para

definir a alocação das tarefas nas máquinas e o roteamento de veículos. O Modelo MILP 2, por sua vez adota uma abordagem indexada no tempo, baseada na discretização do horizonte temporal, em que as variáveis de decisão indicam os momentos exatos de início e término dos processos de cada tarefa, bem como suas respectivas entregas. A Tabela 3.6 apresenta os parâmetros e as variáveis de decisão do Modelo MILP 1.

Tabela 3.6. Parâmetros e variáveis de decisão para o modelo 1 de Programação Linear Inteira Mista (MILP)

Indices	
i	Máquinas.
j,k	Tarefas.
v	Veículos.
O	Origem (Deposito).
Conjuntos	
$M = \{1,2,\ldots,m\}$	Conjunto contendo m máquinas.
$I = \{1,2,\ldots,n\}$	Conjunto contendo n tarefas.
$N = I \cup \{O\}$	Conjunto contendo tarefas e a uma tarefa fictícia da origem.
$L = \{1,2,\ldots,l\}$	Conjunto contendo l Veículos.
Parametros	
p_{ij}	Tempo de processamento da tarefa j na máquina i.
s_{ijk}	Tempo de preparação para a tarefa k após a tarefa j na máquina i.
t_{jk}	Tempo de viagem entre o local da tarefa j e o local da tarefa k.
w_j	Penalidade por atraso da tarefa j.
d_j	Data de entrega da tarefa j.
h_j	Demanda do cliente j (tamanho da tarefa j).
q_v	Capacidade do veículo v.
Variáveis de decisão	
Z_{vjk}	1 - Se o veículo v for usado para viajar entre os pontos j e k. 0 - Caso contrário.
X_{ijk}	1 - Se a tarefa j precede a tarefa k na máquina i, 0 - Caso contrário.
Y_{ij}	1 - Se a tarefa j é processada na máquina i, 0 - Caso contrário.
U_v	1 - Se o veículo v é usado para fazer entregas. 0 - Caso contrário.
C_j	Tempo de conclusão da tarefa j.
S_v	Tempo de início da viagem do veículo v.
T_j	Atraso na entrega da tarefa j.
D_j	Data da entrega da tarefa j.

Fonte: Próprio Autor

O modelo também utiliza os seguintes parâmetros.

- MG é um valor suficientemente grande definido por $\sum_{i\in M}\sum_{j\in N}\sum_{k\in N} t_{jk}+p_{jk}+s_{ijk}$.
- $N = I \cup \{O\}$, representa um conjunto contendo os trabalhos e um trabalho fictício começando na origem.

A seguir, apresenta-se a formulação do modelo matemático. O objetivo do modelo é:

$$\min \ \text{TWT} = \sum_{j\in I} w_j T_j \tag{3.21}$$

Sujeito a:

$$\sum_{v\in L}\sum_{k\in N,k\neq j} Z_{vjk} = 1, \quad \forall j \in I \tag{3.22}$$

$$\sum_{j\in N}\sum_{k\in N,k\neq j} Z_{vjk} h_j \leq q_v U_v, \quad \forall v \in L \tag{3.23}$$

$$\sum_{k\in N} Z_{v0k} = U_v, \quad \forall v \in L \tag{3.24}$$

$$\sum_{j\in N} Z_{vjh} = \sum_{k\in N} Z_{vhk} \forall h \in N, \quad \forall v \in L \tag{3.25}$$

$$\sum_{k\in I} X_{i0k} \leq 1, \quad \forall i \in M \tag{3.26}$$

$$\sum_{i\in M} Y_{ij} = 1, \quad \forall j \in I \tag{3.27}$$

$$Y_{ij} = \sum_{k\in N,k\neq j} X_{ijk}, \quad \forall i \in M, \forall j \in I \tag{3.28}$$

$$Y_{ik} = \sum_{j\in N,k\neq j} X_{ijk}, \quad \forall i \in M, \forall k \in I \tag{3.29}$$

$$C_k - C_j + MG(1 - X_{ijk}) \geq s_{ijk} + p_{ik}, \quad \forall j \in N, \forall k \in I, j \neq k, \forall i \in M \tag{3.30}$$

$$S_v \geq C_k - MG(1 - \sum_{j \in N, j \neq k} Z_{vjk}), \quad \forall v \in L, \forall k \in N \tag{3.31}$$

$$D_k - S_v \geq t_{0k} - MG(1 - Z_{v0k}), \forall v \in L, \qquad \forall k \in N \tag{3.32}$$

$$D_k - D_j \geq t_{jk} - MG(1 - \sum_{v \in L} Z_{vjk}), \quad \forall j \in I, \forall k \in N, j \neq k \tag{3.33}$$

$$T_j \geq D_j - d_j, \quad \forall j \in I \tag{3.34}$$

$$Z_{vjk} \in 0, 1, \quad \forall j \in N, k \in N, j \neq k, \quad v \in L \tag{3.35}$$

$$X_{ijk} \in \{0, 1\} \quad \forall j \in N, \forall k \in N, j \neq k, \quad \forall i \in M \tag{3.36}$$

$$Y_{ij} \in \{0, 1\}, \quad \forall j \in I, \quad \forall i \in M \tag{3.37}$$

$$U_v \in \{0, 1\} \quad \forall v \in L \tag{3.38}$$

$$C_j \geq 0 \quad \forall j \in I \tag{3.39}$$

$$S_v \geq 0 \quad \forall v \in L \tag{3.40}$$

$$T_j \geq 0 \quad \forall j \in I \tag{3.41}$$

$$D_j \geq 0 \quad \forall j \in I \tag{3.42}$$

A função objetivo, definida pela Equação (3.21), visa minimizar a soma dos atrasos ponderados das tarefas. As condições descritas em (3.22) garantem que

cada cliente seja atendido exatamente uma vez e por um único veículo. As restrições (3.23) garantem que as capacidades dos veículos sejam respeitadas, considerando $h_0 = 0$. As condições (3.24) e (3.25) garantem que, quando utilizado, um veículo inicie e termine sua rota no depósito. As restrições (3.26) garantem que apenas uma tarefa seja executada após a tarefa fictícia I_0 em cada máquina. As restrições (3.27) garantem que cada tarefa seja executada em apenas uma máquina. As restrições (3.28) e (3.29) determinam que cada tarefa tenha precisamente uma tarefa sucessora e uma tarefa predecessora. As restrições (3.30) definem o tempo de conclusão de cada tarefa, considerando $C_0 = 0$. As restrições (3.31) especificam o horário de início da viagem de cada veículo. As restrições (3.32) e (3.33) determinam os tempos de chegada dos veículos. Por último, as restrições (3.34) estabelecem atrasos nas tarefas com base nas suas datas de vencimento. As restrições (3.35, 3.36, 3.37, 3.38, 3.39, 3.40, 3.41, 3.42) definem a natureza das variáveis.

A Tabela 3.7 apresenta os parâmetros e as variáveis de decisão do Modelo MILP 2.

Tabela 3.7. Parâmetros e variáveis de decisão para o modelo 2 de Programação Linear Inteira Mista (MILP)

Índices	
i	Máquinas.
j,k	Tarefas.
v	Veículos.
O	Origem (Deposito).
Conjuntos	
$M = \{1,2,\ldots,m\}$	Conjunto contendo m máquinas.
$I = \{1,2,\ldots,n\}$	Conjunto contendo n tarefas.
$N = I \cup \{O\}$	Conjunto contendo tarefas e a uma tarefa fictícia da origem.
$L = \{1,2,\ldots,l\}$	Conjunto contendo l Veículos.
Parametros	
p_{ij}	Tempo de processamento da tarefa j na máquina i.
s_{ijk}	Tempo de preparação para a tarefa k após a tarefa j na máquina i.
t_{jk}	Tempo de viagem entre o local j e o local k.
w_j	Penalidade por atraso da tarefa j.
d_j	Data de entrega da tarefa j.
h_j	Demanda do cliente j (tamanho da tarefa j).
q_v	Capacidade do veículo v.
Variáveis de decisão	
X_{ijt}	1 - Se a tarefa j inicia no tempo t na máquina i, 0 - Caso contrário.
Y_{vjt}	1 - Se a tarefa j inicia a viagem no tempo t no veículo v, 0 - Caso contrário.
S_{vj}	1 - Se a tarefa j é entregue pelo veículo v. 0 - Caso contrário.
C_v	Tempo de início da viagem do veículo. 0 - Caso contrário.

Fonte: Próprio Autor

No modelo 2, utiliza-se também os seguintes parâmetros:

- H - Horizonte de tempo para o sequenciamento das tarefas.
- H_v - Horizonte de tempo para o roteamento de veículos.
- $\tau = \{1,2,\ldots,b\}$ - O conjunto de períodos de tempos, indexado $t = 1, 2, ..., b$.
- $\tau' = \{1,2,\ldots,b'\}$ - O conjunto de períodos de tempos, indexado $t = 1, 2, ..., b'$.
- T_{jb} - Penalidade de atraso para cada tarefa.

O Modelo Matemático 2 é apresentado a seguir:

$$\min \sum_{v \in L} \sum_{j \in I} \sum_{b \in \{0..H_v\}} Y_{vjt} * T_{jb} \tag{3.43}$$

sujeito a:

$$\sum_{i \in M} \sum_{b \in \{0..H-p_{ij}+1\}} X_{ijb} = 1, \quad \forall j \in I \tag{3.44}$$

$$\begin{gathered} X_{ijb} + \sum_{g \in \{max(0,b-p_{ik}-s_{ikj}+1)..b+p_{ij}+s_{ijk}-1\}} X_{ikg} \leq 1, \\ \forall j,k \in I, k \neq j, i \in M, b \in \{0..H - p_{ij} - s_{i,j.k} + 1\} \end{gathered} \tag{3.45}$$

$$\sum_{v \in L} \sum_{b \in \{0..H_v\}} Y_{vjb} = 1, \quad \forall j \in I \tag{3.46}$$

$$\begin{gathered} \sum_{g \in \{max(0,b-t_{kj}+1)..min(b+t_{jk}-1,H_v)\}} Y_{vkg} \leq 1 \\ \forall j,k \in I j \neq k, v \in L, b \in \{0..H_v\} \end{gathered} \tag{3.47}$$

$$\sum_{i \in M} \sum_{b \in \{0..H\}} (b + p_{ij}) * X_{ijb} \leq C_v + H_v * (1 - S_{vj}), \quad \forall j \in I, v \in L \tag{3.48}$$

$$\sum_{b \in \{0..H_v\}} b * Y_{vjb} \geq C_v + t_{0j} - H_v * (1 - S_{vj}), \quad \forall j \in I, v \in L \tag{3.49}$$

$$\sum_{j \in I} \sum_{b \in \{0..H_v\}} Y_{vjb} * h_j \leq q_v, \quad \forall v \in L \tag{3.50}$$

$$\sum_{v \in L} S_{vj} = 1, \quad \forall j \in I \tag{3.51}$$

$$X_{ijt} \in \{0,1\}, \quad \forall i \in M, \quad \forall j \in I, \quad \forall t \in H \tag{3.52}$$

$$Y_{vjt} \in \{0,1\}, \quad \forall v \in L, \quad \forall j \in N, \quad \forall t \in H_v \tag{3.53}$$

$$S_{vj} \in \{0,1\}, \quad \forall v \in L, \quad \forall j \in I \tag{3.54}$$

$$C_v \geq 0, \quad \forall v \in L \tag{3.55}$$

A função objetivo, definida pela Equação (3.43), visa minimizar a soma dos atrasos totais ponderados das tarefas. As restrições (3.44) garantem que cada tarefa seja executada uma única vez, em uma máquina. As restrições (3.45) garantem o sequenciamento das tarefas nas diferentes máquinas, respeitando os tempos de processamento e os tempos de preparação. As restrições (3.46) garantem que cada tarefa seja entregue por um único veículo. As restrições (3.47) garantem a ordem de entrega das tarefas nos veículos. As restrições (3.48) determinam o tempo de início da viagem de cada veículo, respeitando os tempos de conclusão das tarefas. As restrições (3.49) definem as datas de entrega. As restrições (3.50) garantem que as capacidades dos veículos sejam respeitadas, e por fim, as restrições (3.51) garantem a entrega de cada tarefa por um único veículo. As restrições (3.52, 3.53, 3.54, 3.55) definem a natureza das variáveis.

Capítulo 4

Métodos para Resolução de Problemas de Otimização Combinatória

Este capítulo explora métodos avançados de resolução de problemas de otimização combinatória, fundamentais para enfrentar desafios complexos em diversas aplicações industriais e logísticas. A eficiência e a eficácia desses métodos são cruciais para aprimorar o desempenho dos sistemas de produção e distribuição.

Inicialmente, apresentamos a representação da solução, uma etapa fundamental que define como as soluções para o problema são codificadas e manipuladas pelos algoritmos. Uma representação adequada facilita a aplicação de métodos de otimização e a exploração eficiente do espaço de busca.

Em seguida, discutimos os algoritmos de decodificação, que transformam a representação da solução em planos de ação concretos. Esses algoritmos são fundamentais para avaliar a qualidade das soluções geradas e orientar o processo de otimização. O capítulo prossegue com a apresentação dos algoritmos construtivos, que constroem soluções iniciais para o problema. Essas soluções servem como ponto de partida para algoritmos de melhoria subsequentes.

Exploramos também as heurísticas de busca de vizinhança, técnicas que refinam soluções existentes ao explorar suas vizinhanças. Essas heurísticas são eficazes para encontrar ótimos locais e soluções de maior qualidade.

Um destaque deste capítulo é o *framework* que utiliza aprendizado de máquina para selecionar a melhor heurística de busca de vizinhança para uma determinada instância do problema. Esse *framework* considera as características estatísticas da instância do problema para escolher a estratégia de busca mais adequada, melhorando significativamente a qualidade da solução.

Além disso, apresentamos o algoritmo PPO, uma técnica de aprendizado por reforço que mostrou resultados promissores na resolução de problemas de otimização combinatória. O PPO ajusta políticas de decisão com base em *feedback* contínuo, facilitando a adaptação a diferentes cenários e requisitos do problema.

O capítulo inclui uma análise da meta-heurística RVND, que explora diversas vizinhanças de forma aleatória para melhorar a solução corrente. Esta técnica é conhecida por sua robustez e por sua capacidade de evitar que o algoritmo fique preso em mínimos locais.

Por fim, introduzimos uma abordagem híbrida, PPO-VND, que integra o VND com o algoritmo de aprendizado por reforço profundo PPO. Esta abordagem combina as forças de ambos os métodos, utilizando o DRL para guiar a busca de vizinhança do VND de maneira adaptativa e eficiente.

4.1 Representação da Solução e Algoritmos de Decodificação da Solução

A representação da solução do problema divide-se em duas partes principais: o sequenciamento das tarefas nas máquinas e o roteamento das entregas aos clientes. A primeira parte é uma matriz de números inteiros s de tamanho $m \times n$, onde m representa o número de máquinas e n, o número de tarefas. Cada linha da matriz s corresponde a uma máquina e indica a ordem de sequenciamento das tarefas nessa máquina.

A segunda parte da solução é uma matriz de números inteiros r de tamanho $l \times n$, onde l representa o número de veículos. Cada linha da matriz r corresponde a um veículo e define a ordem de entrega das tarefas aos clientes. A partir das duas matrizes (s e r), é possível obter uma solução completa (*solução*) para o problema, permitindo o cálculo da soma dos atrasos totais ponderados das tarefas.

A *solução* completa ($\{s, r\}$) é construída a partir de duas listas de tarefas, $L1$ e $L2$. A lista $L1$ representa a ordem de inserção das tarefas no sequenciamento s utilizando o algoritmo de decodificação NEH (*Nawaz-Enscore-Ham*), que minimiza o tempo total de processamento ao inserir as tarefas sequencialmente nas máquinas. A lista $L2$ define a ordem de inserção das tarefas nas rotas dos veículos r por meio do algoritmo de decodificação PIFH (*Push Forward Insertion Heuristic*), que otimiza o atraso total ponderado das tarefas ao alocá-las sequencialmente nas rotas dos veículos. A Figura 4.1 ilustra o processo de decodificação de uma solução a partir das listas de tarefas $L1$ e $L2$.

Essa estrutura modular permite integrar de forma eficaz a programação das tarefas nas máquinas e o roteamento das entregas aos clientes, resultando em uma abordagem otimizada tanto para o processamento nas máquinas quanto para a distribuição logística.

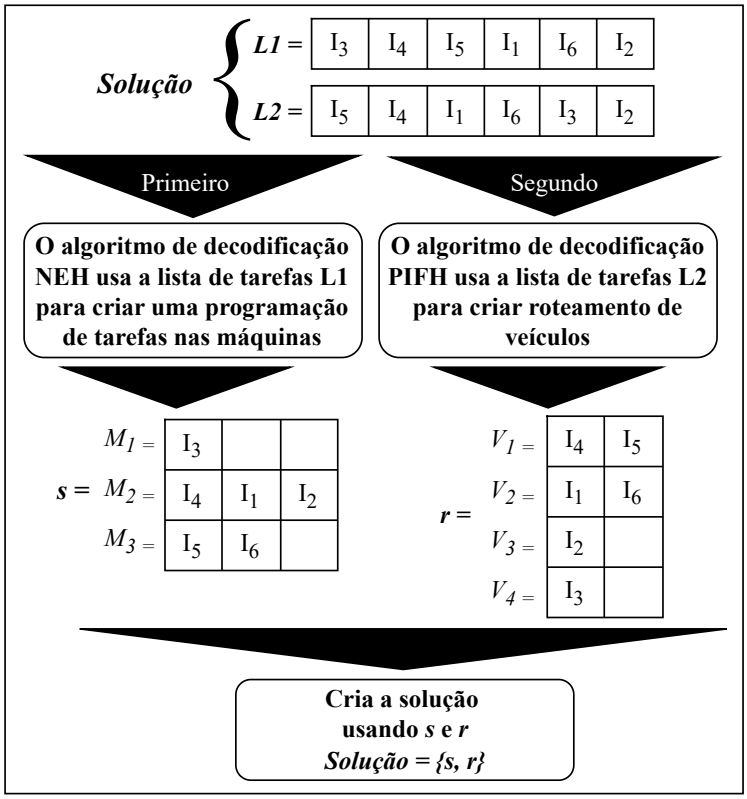


Figura 4.1. Processo de decodificação da solução.

Fonte: Próprio Autor

A heurística NEH, proposta por Nawaz et al. (1983), é um algoritmo guloso amplamente utilizado para resolver problemas de sequenciamento de tarefas em máquinas. Com base nessa heurística, o algoritmo de decodificação NEH é empregado para construir sequências de tarefas nas máquinas eficientemente.

O algoritmo de decodificação NEH insere sequencialmente as tarefas da lista $L1$ nas máquinas, visando minimizar o tempo de conclusão (*completion time*) em cada máquina. O processo de inserção das tarefas é realizado de forma a encontrar a melhor posição possível para cada uma, garantindo que o tempo de conclusão seja o menor possível após cada inserção.

Inicialmente, no algoritmo de decodificação NEH, a matriz de sequenciamento s é vazia, e as tarefas da lista $L1$ são inseridas uma a uma em s. Para cada tarefa $i \in L1$, o algoritmo verifica todas as possíveis posições no sequenciamento s e insere a tarefa na posição que resulta no menor tempo de conclusão. Esse procedimento é repetido até que todas as tarefas de $L1$ sejam inseridas no sequenciamento s. O algoritmo de decodificação NEH é detalhado no Algoritmo 1.

Algorithm 1 Algoritmo de Decodificação NEH.

```
1: função NEH(L1)
2:     s ← ∅                    ▷ matriz de sequenciamento das tarefas nas máquinas
3:     para i ← 1 até n faça
4:         Insere a tarefa L1[i] na melhor posição no sequenciamento s
5:     fim para
6:     retorna s
7: fim função
```

Esse processo garante que o tempo de conclusão das tarefas seja minimizado iterativa e eficientemente, aproveitando a natureza gulosa da heurística NEH. O algoritmo é particularmente eficaz em problemas de sequenciamento de tarefas em máquinas, onde a minimização do tempo de conclusão é crucial para a eficiência operacional, entre tanto seu tempo de execução é muito longo para instâncias de grandes porte Ta et al. (2015). Em resumo, o algoritmo de decodificação NEH é uma ferramenta poderosa para o sequenciamento de tarefas em ambientes de máquinas paralelas, proporcionando soluções que minimizam o tempo total de processamento de forma eficiente e sistemática.

A heurística PIFH, proposta por Solomon (1987), é um algoritmo eficiente para a construção de rotas de entrega, utilizando um critério guloso para inserir tarefas sequencialmente nas rotas. Com base nesta heurística, o algoritmo de decodificação PIFH é utilizado para otimizar o roteamento das tarefas, minimizando o atraso total ponderado (TWT, do inglês *Total Weighted Tardiness*).

O algoritmo de decodificação PIFH insere sequencialmente as tarefas da lista $L2$ nas rotas dos veículos. A cada iteração, uma tarefa é inserida na posição que minimiza o atraso total ponderado nas rotas existentes. Esse processo é repetido até que todas as tarefas sejam alocadas nas rotas dos veículos, resultando na

definição final das rotas (r). Inicialmente, a matriz de rotas r está vazia. A cada iteração, a tarefa $i \in L2$ é inserida na posição que minimiza o atraso total ponderado. Esse método garante que, a cada inserção, o impacto no desempenho global da rota seja minimizado, otimizando assim o roteamento das tarefas de forma eficiente. O algoritmo de decodificação PIFH é apresentado no Algoritmo 2.

Algorithm 2 Algoritmo de Decodificação PIFH.

```
1: função PIFH( L2, s )
2:     r ← ∅                                   ▷ Matriz das rotas de veículos
3:     para i ← 1 até n faça
4:         Insere tarefa L2[i] na melhor posição nas rotas r
5:     fim para
6:     retorna r
7: fim função
```

Este procedimento permite a construção eficiente de rotas de entrega, focando na minimização do atraso Reina (2012). A heurística PIFH é particularmente eficaz em cenários onde o atraso total ponderado é uma métrica crítica para o desempenho logístico, entretanto o seu tempo de execução é muito longo para instâncias grandes. Em resumo, o algoritmo de decodificação PIFH oferece uma abordagem sistemática e eficiente para o problema de roteamento de veículos, garantindo que as rotas sejam otimizadas em termos de atraso total ponderado. O método é essencial para melhorar a eficiência operacional em ambientes de distribuição.

4.2 Solução Inicial

Para criar uma solução inicial para o problema ($solução = s,r$), é necessário seguir uma série de etapas bem definidas, envolvendo a construção das listas de tarefas ($L1$ e $L2$) e a aplicação de algoritmos de decodificação para definir o sequenciamento das tarefas nas máquinas (s) e as rotas de entrega (r). A seguir, descrevemos detalhadamente o método proposto.

A primeira etapa do algoritmo consiste na construção da lista de tarefas $L1$. A lista $L1$ é construída pela ordenação das tarefas em ordem decrescente com

base na soma dos tempos médios de processamento e preparação. Essa ordenação garante que as tarefas mais críticas em termos de tempo de processamento e preparação sejam priorizadas no sequenciamento.

Na segunda etapa, define-se o sequenciamento das tarefas nas máquinas (s). Uma vez construída a lista $L1$, utilizamos o algoritmo de decodificação NEH para definir o sequenciamento das tarefas nas máquinas, resultando na matriz de sequenciamento s.

Em seguida, procede-se à construção da lista de tarefas $L2$. A partir do sequenciamento s, a lista $L2$ é criada utilizando a regra da data de entrega mais próxima (EDD, do inglês *Earliest Due Date*). As tarefas em $L2$ são ordenadas de forma crescente, com base na diferença entre a data de entrega (d_j) e o tempo de conclusão da tarefa (C_j), priorizando aquelas que precisam ser entregues mais rapidamente em relação às suas datas de entrega. A próxima etapa do algoritmo consiste na criação das rotas de entrega (r) a partir da lista $L2$ utilizando o algoritmo de decodificação PIFH, resultando em r.

A última etapa do algoritmo consiste em definir a solução inicial do problema, combinando as duas partes obtidas: o sequenciamento das tarefas nas máquinas (s) e as rotas de entrega (r). O Algoritmo 3 apresenta o pseudocódigo da solução inicial, enquanto a Figura 4.2 ilustra as etapas de construção dessa solução.

Algorithm 3 Solução Inicial.

```
1: função SOLUÇÃO INICIAL( )
2:     Criar lista de tarefas L1
3:     s ← NEH(L1)
4:     Criar lista de tarefas L2 a partir do sequenciamento s
5:     r ← PIFH(L2, s)
6:     solução ← {s, r}
7:     retorna solução
8: fim função
```

Essa abordagem modular e sistemática permite a criação de uma solução inicial que integra eficientemente o sequenciamento de tarefas nas máquinas e o roteamento das entregas aos clientes. Ao utilizar heurísticas bem estabelecidas como o NEH e o PIFH, conseguimos melhorar tanto o tempo de processamento

quanto o atraso total ponderado das tarefas, estabelecendo uma base sólida para otimizações subsequentes.

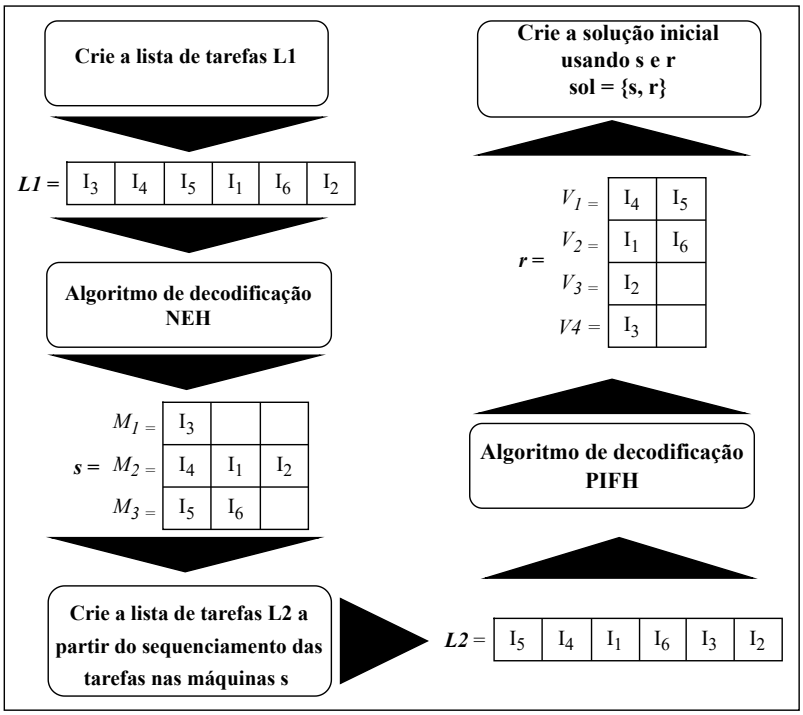


Figura 4.2. Processo da construção da solução inicial.

Fonte: Próprio Autor

4.3 Heurísticas de Busca de Vizinhança

Para implementar as Heurísticas de Busca de Vizinhança, utilizamos dois movimentos fundamentais: troca (*swap*) e inserção (*insertion*). Esses movimentos são cruciais para explorar diferentes soluções no espaço de busca e melhorar iterativamente a solução.

O Algoritmo de Troca altera as posições de dois elementos em uma lista, gerando uma nova sequência. Esse movimento permite permutar duas tarefas, alterando suas posições na sequência. O pseudocódigo do Algoritmo de Troca é apresentado no Algoritmo 4. Nele, os elementos nas posições i e j da lista (*list*)

são trocados de posição utilizando uma variável auxiliar. Embora simples, esse movimento é poderoso, pois permite testar novas sequências de tarefas que podem levar a uma solução melhor.

O Algoritmo de Inserção, por sua vez, envolve a remoção de um elemento da lista e sua reinserção em uma posição diferente, reorganizando a sequência das tarefas. Esse movimento permite melhorar o critério de otimização ao considerar diferentes ordenações. O pseudocódigo do Algoritmo de Inserção é apresentado no Algoritmo 5. Nele, o elemento na posição i é removido da lista e inserido na posição j da lista. Esse movimento é útil para testar diversas posições de tarefas na lista, oferecendo maior flexibilidade para encontrar uma sequência que conduza a uma solução melhor.

Os movimentos de Troca e Inserção são amplamente utilizados nas Heurísticas de Busca de Vizinhança para explorar o espaço de soluções vizinhas. Aplicando esses movimentos iterativamente, o algoritmo pode explorar a vizinhança local e, em alguns casos, identificar soluções de melhor qualidade. A seguir, detalharemos as Heurísticas de Busca de Vizinhança que utilizam esses movimentos e abordagens complementares para resolver o problema proposto.

Algorithm 4 Algoritmo de troca.

```
1: função TROCA( list, i, j )
2:     aux ← list[i]
3:     list[i] ← list[j]
4:     list[j] ← aux
5:     retorna list
6: fim função
```

Algorithm 5 Algoritmo de Inserção.

```
1: função INSERÇÃO( list, i, j )
2:     aux ← list[i]
3:     list.remove(aux)
4:     list.insere(j, aux)
5:     retorna list
6: fim função
```

Para melhorar a qualidade das soluções, desenvolvemos a Heurística de Busca de Vizinhança (NSH, do inglês *Neighborhood Search Heuristics*). A NSH opera iterativamente, aplicando os movimentos de troca ou inserção nas listas de tarefas $L1$ ou $L2$ para explorar o espaço de soluções vizinhas. O algoritmo finaliza retornando a melhor solução encontrada durante sua execução. O Algoritmo 6 apresenta o pseudocódigo da NSH.

A NSH utiliza como entrada duas listas de tarefas $L1$ e $L2$ e conjunto de parâmetros que determinam seu comportamento. O parâmetro *movimento* especifica se o algoritmo usará movimentos de troca ou inserção, enquanto o parâmetro *lista* define qual lista de tarefas ($L1$ ou $L2$) será manipulada. O parâmetro *restart* determina se a busca deve reiniciar ao final, caso uma solução melhor que a inicial seja descoberta.

A Tabela 4.1 apresenta as oito variações de Heurísticas de Busca de Vizinhança, denominadas NSH 1 a NSH 8, identificando os parâmetros específicos de cada variação.

Tabela 4.1. Nome das Heurísticas de Busca de Vizinhança e Parâmetros.

Nome	Parâmetros		
	Movimentos	*Lista*	*Restart*
NSH 1	Trocar	L2	*False*
NSH 2	Trocar	L2	*True*
NSH 3	Inserção	L2	*False*
NSH 4	Inserção	L2	*True*
NSH 5	Trocar	L1	*False*
NSH 6	Trocar	L1	*True*
NSH 7	Inserção	L1	*False*
NSH 8	Inserção	L1	*True*

Fonte: Próprio Autor

O Algoritmo NSH (Algoritmo 6) inicia criando uma solução inicial do problema com base nas listas de tarefas $L1$ e $L2$. Dependendo dos parâmetros *lista* e *movimento*, o algoritmo aplica movimentos de troca ou inserção para gerar novas soluções. Se os movimentos (troca ou inserção) forem aplicados à lista de tarefas $L1$, o algoritmo de decodificação NEH gera um novo sequenciamento das tarefas nas máquinas (s'). Esse sequenciamento (s') é então utilizado para gerar a lista

de tarefas ($L2'$). O algoritmo PIFH, por sua vez, utiliza $L2'$ para definir as rotas de entrega das tarefas (r'). Por fim, s' e r' compõem a solução ($solução'$) para o problema.

Alternativamente, se os movimentos forem aplicados à lista de tarefas $L2$, o sequenciamento inicial das tarefas nas máquinas (s) é mantido, e as rotas de entrega das tarefas (r') são definidas pelo algoritmo PIFH. Nesse caso, s e r' formam a nova solução ($solução'$). A cada iteração do algoritmo, a solução atual é comparada com a melhor solução encontrada até o momento. Caso a solução atual seja superior, a melhor solução é atualizada.

No final do algoritmo, o parâmetro *restart* determina se a busca deve ser reiniciada caso uma solução superior à solução inicial seja encontrada. Por fim, o algoritmo retorna a melhor solução encontrada, juntamente com as listas $L1$ e $L2$ que deram origem a essa solução.

4.4 *Framework*

Para lidar eficazmente com problemas de otimização combinatória, é fundamental identificar o algoritmo que oferece os melhores resultados de forma consistente para um conjunto específico de instâncias. Embora essa abordagem tradicional possa produzir resultados impressionantes em alguns casos, ela pode falhar em outros devido à grande variabilidade de cenários presentes em um conjunto de instâncias.

Para superar essa limitação, desenvolvemos um *Framework* que utiliza técnicas de aprendizado de máquina (ML, do inglês *Machine Learning*) para identificar a heurística mais eficaz para cada instância do problema com base em suas características estatísticas. Este *Framework* não apenas seleciona a melhor NSH, mas também se adapta dinamicamente às características das instâncias do problema, garantindo uma abordagem mais robusta e eficiente. Ao analisar as características de cada instância e utilizar um modelo preditivo treinado com uma base de dados, o *Framework* é capaz de recomendar a NSH mais adequada para resolver cada instância do problema.

A seguir, detalharemos a estrutura e o funcionamento do *Framework*, incluindo a seleção da base de dados, o treinamento do modelo de aprendizado,

Algorithm 6 Heurística de Busca de Vizinhança.

```
1: função NSH( L1, L2, movimentos, lista, restart )
2:     NEH e PIFH determinam a solução = {s,r} a partir da lista L1 e L2.
3:     melhor_L1 ← L1
4:     melhor_L2 ← L2
5:     se lista ='L1' então                ▷ O valor 'L1' indica o uso da Lista L1.
6:         L ← L1
7:     senão
8:         L ← L2
9:     fim se
10:    i ← 0
11:    enquanto i < len(L) faça
12:        Melhor_FO =F(solução)
13:        j ← i + 1
14:        enquanto j ≤ len(L) faça
15:            se movimentos = TROCAR então
16:                L' ← trocar(L, i, j)
17:            senão
18:                L' ← inserção(L, i, j)
19:            fim se
20:            se lista ='L1' então
21:                s' ← NEH (L')
22:                Criar lista de tarefas L2' a partir de s'
23:                r' ← PIFH (s', L2')
24:                solução' ← {s',r'}
25:            senão
26:                r' ← PIFH (s, L')
27:                solução' ← {s,r'}
28:            fim se
29:            se F(solução') <F(solução) então
30:                solução ← solução'
31:                se lista ='L1' então
32:                    Melhor_L1 ← L
33:                    Melhor_L2 ← L2'
34:                senão
35:                    Melhor_L1 ← L1
36:                    Melhor_L2 ← L
37:                fim se
38:            fim se
39:            j+ = 1
40:        fim enquanto
41:        i+ = 1
42:        se restart =True E i = len(L) E Melhor_FO <F(solução) então
43:            L1 ← Melhor_L1
44:            L2 ← Melhor_L2
45:            i ← 0
46:        fim se
47:    fim enquanto
48:    retorna solução, Melhor_L1, Melhor_L2
49: fim função
```

e sua aplicação prática na seleção de heurísticas para problemas de otimização combinatória.

Inicialmente, utilizamos um algoritmo de clusterização para agrupar instâncias com base em características semelhantes. Neste trabalho, aplicamos o algoritmo *k-means*, um método de particionamento que divide o conjunto de dados em k *clusters*, atribuindo cada instância ao *cluster* cujo centroide está mais próximo. O algoritmo ajusta iterativamente os centroides dos *clusters* até que a variação dentro dos *clusters* seja minimizada, resultando em grupos de instâncias com características semelhantes.

Após a clusterização, para cada grupo, foi escolhida uma instância que melhor representa o grupo, selecionada como a instância mais próxima do centroide do *cluster*, garantindo uma amostra representativa de cada grupo. Essas instâncias foram utilizadas para compor o conjunto de treinamento.

Em seguida, para cada instância do conjunto de treinamento, foram aplicadas as oito NSH. A NSH que produziu a melhor solução para cada instância foi selecionada para representar aquele grupo específico.

Com base nos padrões e regras identificados no conjunto de treinamento, desenvolvemos um modelo de ML para determinar a melhor heurística para resolver uma instância específica. O modelo de ML leva em consideração fatores como: número de tarefas (n), número de máquinas (m), tempos médios de processamento ($\overline{p}$), configuração ($\overline{s}$), e viajem ($\overline{t}$), além da demanda média ($\overline{h}$) e da capacidade média do veículo ($\overline{q}$). Esses fatores são utilizados como entradas para o modelo, que prevê a NSH mais adequada para resolver a instância.

O objetivo do treinamento do modelo de ML é construir uma função $h(x) : x \rightarrow Y$, de modo que $h(x)$ possa prever corretamente o valor correspondente de Y. Após o processo de treinamento, obtemos um preditor $h(x)$ que, com base nas características da instância, retorna a melhor NSH para resolvê-la. O esquema do *Framework* está ilustrado na Figura 4.3.

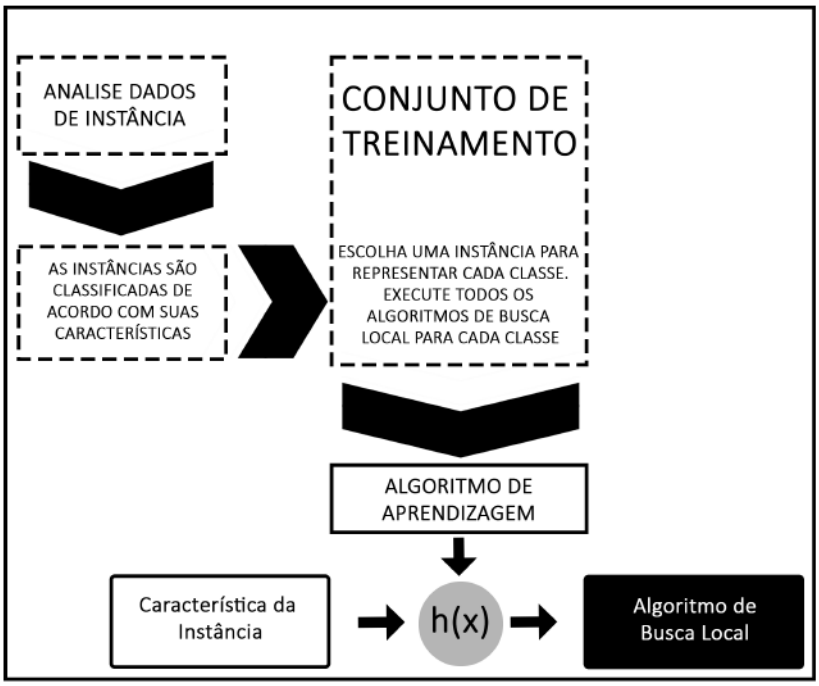


Figura 4.3. Esquema de execução do Framework.

Fonte: Próprio Autor

O preditor do nosso *Framework* é uma rede neural do tipo *Multilayer Perceptron* (MLP), com várias camadas de neurônios. Cada camada realiza uma combinação linear das entradas para calcular sua saída. O treinamento do MLP foi realizado utilizando o Algoritmo Adam, conforme descrito por Kingma & Ba (2014). Esse algoritmo é uma técnica de otimização baseada em gradiente de primeira ordem, que utiliza estimativas adaptativas de momentos para ajustar os pesos dos neurônios de maneira eficiente.

Para explicar de forma mais acessível, o MLP é uma rede neural com várias camadas, onde cada camada processa a informação recebida da camada anterior, realizando cálculos para gerar uma saída. O Algoritmo Adam facilita o aprendizado da rede ajustando os pesos dos neurônios de maneira precisa e rápida, garantindo uma otimização eficaz.

O desempenho da rede neural é avaliado usando o Erro Quadrático Médio (MSE, do inglês *Mean Squared Error*). O MSE calcula a diferença entre os valores previstos pela rede e os valores reais esperados, fornecendo uma medida quantita-

tiva da precisão das previsões. Quanto menor o MSE, melhor o desempenho da rede, indicando que as previsões são mais precisas.

Em resumo, o uso do MLP treinado com o Algoritmo Adam, juntamente com a avaliação por MSE, permite a construção de um modelo robusto e preciso para previsões. Isso destaca a capacidade da rede neural de aprender e generalizar a partir dos dados fornecidos, proporcionando um *Framework* poderoso para otimização combinatória.

4.5 Proximal Policy Optimization

Problemas sequenciais de tomada de decisão são comumente resolvidos com RL, no qual um agente interage com um ambiente estocástico e aprende a mapear estados em ações para maximizar recompensas cumulativas. O problema de RL é modelado como um Processo de Decisão de Markov (MDP, do inglês *Markov Decision Process*), que é uma tupla $\{S, A, p, r(s, a)\}$. Nesse contexto, S representa o conjunto de estados, A o conjunto de ações, p é a função de transição que denota as probabilidades de mudança de estado (entre 0 e 1), e a função de recompensa $r(s, a)$ atribui um valor numérico ao par estado-ação. O objetivo principal do agente é encontrar uma política ótima que associe cada estado à ação mais adequada, maximizando a soma das recompensas ao longo do tempo (Alves & Mateus, 2020).

O PPO, introduzido por Schulman et al. (2017), é um algoritmo de RL baseado no conceito de Gradientes de Política (PG, do inglês *Policy Gradients*), que visa otimizar as políticas dos agentes usando uma abordagem *on-policy*. O PPO utiliza uma Rede Neural Profunda (DNN, do inglês *Deep Neural Network*) para mapear ações a estados, permitindo que os agentes aprendam diretamente a política ótima enquanto maximizam a recompensa cumulativa no ambiente. Diferentemente de outros métodos que derivam a melhor política a partir de funções de valor, o PPO busca encontra diretamente a política ótima, ajustando-a continuamente para lidar com as dinâmicas do ambiente e garantir estabilidade no processo de aprendizado.

O PPO é adequado para problemas em que decisões são tomadas em sequência, com cada ação impactando diretamente o estado futuro do ambiente. Exem-

plos incluem jogos, controle robótico e navegação de *drones*, onde a tomada de decisão em tempo real é crucial para a execução eficaz. Em tais cenários, o PPO ajusta a política do agente de forma adaptativa, maximizando as recompensas e permitindo que o agente evolua continuamente para responder a novas situações e desafios (Schulman et al., 2017).

Os métodos ator-crítico (*Actor-Critic*) em PG combinam a aprendizagem da política com a aproximação da função de valor, onde o ator aprende a política e o crítico a avalia. No PPO, essa abordagem é aprimorada pela introdução de uma restrição que limita a diferença entre a política nova e a anterior, utilizando uma função de perda penalizada. Essa limitação assegura que as atualizações na política sejam progressivas, evitando mudanças abruptas e garantindo que o agente continue operando de forma eficiente mesmo à medida que novas ações são exploradas.

Minimizar a variação durante o treinamento acelera o processo de aprendizagem, e o PPO equilibra a implementação, parametrização e complexidade da amostra, com o objetivo principal de limitar as atualizações do tamanho do passo. Isso permite renovar a política de forma semelhante à atual, evitando que mudanças extensas causem variações significativas no processo de aprendizagem, o que poderia levar a um desempenho insatisfatório (Alves & Mateus, 2020).

Os autores utilizaram o algoritmo PPO da biblioteca *stable-baselines3*, implementada em *Python* (Raffin et al., 2021). O Algoritmo 7 apresenta o pseudocódigo do PPO, que abrange a inicialização dos parâmetros da política e da função de valor, a execução da política para coletar trajetórias, a estimação das vantagens e as atualizações dos parâmetros por meio de gradientes estocásticos.

Algorithm 7 Proximal Policy Optimization

Inicializar Θ e Φ ▷ parâmetros de política e função de valor, respectivamente
para $i = 0,1,2,...$ **faça**
    para $actor = 1,2,...,N_{actors}$ **faça**
        Executar política $\pi(\Theta)$ para T passos e coletar as trajetórias.
        Calcular estimativas de vantagem $\hat{A}_1, ..., \hat{A}_T$ com base na função de valor V_Φ.
    fim para
    Forme um *lote*, de tamanho NT, com trajetórias e vantagens coletadas.
    para $epoch = 1,2,...,K$ **faça**
        Embaralhar *lote* e dividido em *minilotes*
        for all *minilotes* **faça**
            Atualiza Θ com relação à função objetivo via gradiente estocástico.
            Atualizar Φ wrt erro quadrático médio da função de valor V_Φ.
        fim para
    fim para
fim para

O PPO é amplamente utilizado em diversos domínios por sua simplicidade e robustez, tornando-se uma escolha popular para resolver problemas de otimização em ambientes dinâmicos e complexos. A combinação de estabilidade, eficiência e facilidade de implementação torna o PPO uma ferramenta poderosa de aprendizado por reforço, permitindo que agentes aprendam comportamentos complexos com desempenho consistente.

4.5.1 Estado

No contexto do RL, o estado representa uma configuração específica do ambiente em um dado momento. No MDP, o estado, denotado por S, inclui todas as informações relevantes do ambiente que o agente precisa para tomar decisões informadas. O conjunto de todos os possíveis estados forma o espaço de estados, oferecendo uma visão completa da situação atual, como condições do ambiente, posições dos recursos e outras variáveis críticas que possam afetar as decisões do agente.

Neste trabalho, o estado é representado por um vetor que descreve completamente o ambiente atual com o qual o agente interage. Esse vetor inclui uma

solução do problema, com o número de máquinas (m), o número de tarefas (n), a lista dos tempos de conclusão das tarefas nas máquinas ($C_j, \forall j \in N$), a lista de datas de entrega das tarefas aos clientes ($D_j, \forall j \in N$) e a lista de penalidades por atraso das tarefas ($w_j T_j, \forall j \in N$).

4.5.2 Ação

A ação é a decisão tomada pelo agente em um determinado estado e representa a próxima escolha do agente. No MDP, a ação, denotada por A, é um movimento que o agente pode realizar em um estado específico. O conjunto de todas as possíveis ações define o espaço de ações. Em cada estado, o agente escolhe uma ação com base em sua política atual, que mapeia estados para ações.

A escolha da ação é crucial, pois a transição para o próximo estado influencia diretamente as recompensas acumuladas pelo agente. No contexto do PPO, as ações são selecionadas para maximizar a recompensa acumulada, levando em consideração as probabilidades de mudança de estado fornecidas pela função de transição p. Neste caso, a escolha é a próxima vizinhança explorada pela busca local.

A saída do modelo é uma política ótima de solução, baseada nas listas de tarefas $L1$ e $L2$, ajustadas com base na ação tomada pelo agente. Esse processo permite que o agente aprenda a tomar decisões que maximizem a eficiência e eficácia da busca local, explorando diferentes vizinhanças de forma estratégica.

4.5.3 Função de Recompensa

A função de recompensa é um componente fundamental no RL, atribuindo um valor numérico a cada par estado-ação $r(s,a)$. Esse valor representa a recompensa imediata que o agente recebe ao executar uma ação a em um estado s. A função de recompensa guia o comportamento do agente, incentivando-o a tomar ações que conduzam a estados desejáveis e otimizem seu desempenho ao longo do tempo.

No nosso contexto, a função de recompensa é derivada da função objetivo, conforme a Equação 4.1, e considera os limites inferior (LB) e superior (UB). O LB é calculado conforme a Equação 4.2, enquanto o UB é determinado a partir do cenário mais pessimista, assumindo que a distribuição das tarefas será idêntica

entre máquinas e veículos. Além disso, considera-se que a tarefa j será a última a ser executada na máquina e entregue na rota, e que todas as tarefas terão os tempos de processamento, configuração e viajem mais longos possíveis.

A função de recompensa incentiva o agente a maximizar a soma das recompensas ao longo do tempo, conhecida como recompensa acumulada. Para alcançar esse objetivo, o agente deve aprender uma política ótima que maximize a recompensa esperada em cada estado. Essa abordagem promove um comportamento que otimiza o desempenho ao longo do tempo, guiando o agente a tomar ações que resultem em melhores soluções para o problema de otimização combinatória.

$$\text{Função de Recompensa} = -1 * \frac{(\sum_{j \in I} w_j T_j) - LB)}{(UB - LB) * 100} \tag{4.1}$$

$$LB = \sum_{j \in I} w_j * Max((Min(p_{ij} \forall i \in M) + t_{0j} - d_j), 0) \tag{4.2}$$

4.5.4 Modelo de Ambiente

A interação do agente com o ambiente requer um modelo que represente o problema. No nosso caso, o ambiente modela o problema integrado de planejamento de produção e distribuição. A entrada do modelo consiste em duas listas de tarefas ($L1$ e $L2$), usadas para criar a solução do problema ($solução = \{s,r\}$). A saída do modelo é o valor da função objetivo dessa solução. O objetivo do agente é encontrar uma política que maximize as recompensas cumulativas ao longo do tempo, levando a melhores soluções no contexto do problema integrado.

Em nosso trabalho, foi necessário criar um modelo de ambiente específico para cada grupo de instâncias com n igual a 10, 15 e 30, uma vez que o modelo do ambiente está diretamente relacionado ao número de tarefas. O algoritmo de decodificação NEH insere as tarefas da lista $L1$ sequencialmente na solução, minimizando o tempo de conclusão de cada tarefa em cada iteração. Após definir o sequenciamento das tarefas nas máquinas (s), a lista de tarefas $L2$ é criada. Em seguida, o algoritmo de decodificação PIFH insere as tarefas da lista $L2$ na rota (r), visando minimizar o atraso ponderado da tarefa (wt). Ao final do algoritmo,

a solução completa para o problema é definida e o valor da função objetivo da solução é retornado. O Modelo de Ambiente é apresentado no Algoritmo 8.

Algorithm 8 Modelo de ambiente

1: $solução$ = Nova solução$\{s, r\}$.
2: Classifique a lista de tarefas $L1$ em ordem decrescente da soma dos tempos médios de processamento e configuração.
3: $s \leftarrow$NEH($L1$)
4: Crie a lista de tarefas $L2$ a partir do agendamento de tarefas nas máquinas s (Regra EDD)
5: $r \leftarrow$ PIFH $(L2, s)$
6: Cria $solução$ com s e r
7: **retorna** $f(solução)$

4.6 Random Variable Neighborhood Search

A meta-heurística RVND utiliza diversas heurísticas de busca local para encontrar boas soluções em problemas complexos de otimização combinatória. Esta técnica tem se mostrado eficaz em diversos problemas, incluindo o sequenciamento de máquinas paralelas e o roteamento de veículos. A capacidade do RVND de explorar múltiplas estruturas de vizinhança o torna especialmente adequado para resolver uma ampla gama de problemas de otimização combinatória, incluindo problemas integrados de planejamento e distribuição.

O RVND tem sido amplamente utilizado em problemas de sequenciamento de máquinas paralelas, alocando tarefas de maneira eficiente entre as diferentes máquinas para minimizar o tempo total de processamento ou outras métricas de desempenho. Estudos, como o de Fleszar et al. (2012), demonstraram a eficácia do RVND em encontrar boas soluções para problemas de sequenciamento de tarefas em máquinas paralelas.

No contexto do VRP, o RVND também se mostrou uma ferramenta poderosa. Em pesquisas como a de Penna et al. (2013), o RVND foi combinado com o ILS para resolver uma variante do VRP, o HFVRP (*Heterogeneous Fleet Vehicle Routing Problem*), gerando melhorias significativas na qualidade das soluções.

A utilização do RVND em problemas integrados de planejamento da produção e distribuição se justifica por sua capacidade de combinar diferentes estruturas de vizinhança e técnicas de perturbação de soluções, essenciais para evitar que o algoritmo fique preso em ótimos locais e encontrar soluções de alta qualidade. Estudos demonstram que o RVND pode ser eficaz em problemas integrados, promovendo uma sinergia entre os processos de produção e distribuição.

O RVND é uma meta-heurística não determinística de busca local que explora k estruturas de vizinhança, denotadas por $V = V_1, V_2, ..., V_k$, até que uma solução alcance um ótimo local (Hansen & Mladenović, 2001). O algoritmo começa com uma solução inicial e, a cada iteração, uma Heurística de Busca de Vizinhança u é selecionada aleatoriamente. A NSH u é então aplicada, retornando uma solução ótima local. Se essa nova solução for superior à melhor encontrada até o momento, ela é atualizada. Caso contrário, o algoritmo realiza uma perturbação na solução corrente para escapar de ótimos locais. Esse processo se repete até que o número máximo de iterações seja atingido (n, número de tarefas da instância) ou a solução encontrada seja equivalente ao Limite Inferior. A meta-heurística RVND é descrita no Algoritmo 9.

Algorithm 9 RVND

```
1: solução ← SOLUÇÃOINICIAL()
2: melhor_solução ← solução
3: it ← 0
4: enquanto it < n or f(solução)! = LB faça
5:     u ← Selecione uma NSH aleatoriamente.
6:     solução ← NSH u (solução)
7:     se F(solução) < F(melhor_solução) então
8:         melhor_solução ← solução
9:     senão
10:        perturba a solução corrente
11:    fim se
12:    it ← it + 1
13: fim enquanto
14: retorna melhor_solução
```

A perturbação da solução é essencial na meta-heurística RVND para evitar que o algoritmo fique preso em ótimos locais, limitando a qualidade das soluções

encontradas. Ao introduzir perturbações, o algoritmo consegue explorar um espaço de busca maior, aumentando a probabilidade de encontrar soluções mais próximas do ótimo global.

A perturbação consiste em realizar movimentos de troca e inserção aleatórios nas listas de tarefas. A perturbação ocorre em uma parte das listas de tarefas, $L1$ ou $L2$, escolhida aleatoriamente, onde algumas tarefas são removidas e reinseridas, criando uma nova lista. Em seguida, os algoritmos de decodificação NEH e PIFH utilizam as novas listas ($L1$ e $L2$) para definir uma nova solução, que é então empregada no algoritmo.

A meta-heurística RVND é uma poderosa ferramenta para resolver problemas de otimização combinatória, permitindo uma exploração eficaz e diversificada do espaço de soluções. A utilização de múltiplas estruturas de vizinhança e a introdução de perturbações ajudam a evitar ótimos locais e a melhorar a qualidade das soluções obtidas.

4.7 PPO - VND

A meta-heurística híbrida *Proximal Policy Optimization* com *Variable Neighborhood Descent* (PPO-VND), proposta neste trabalho, combina técnicas de aprendizado por reforço profundo e busca local determinística para resolver problemas complexos de otimização. O objetivo do PPO-VND é melhorar a eficácia e a eficiência dos algoritmos de busca local, utilizando a capacidade de aprendizado do PPO para prever e selecionar, em cada iteração, as heurísticas de busca de vizinhança mais apropriadas para aquele momento.

A meta-heurística híbrida PPO-VND é apresentada no Algoritmo 10. O algoritmo inicia com uma solução inicial, definida como a melhor solução ($melhor\_solução$). Em seguida, o estado inicial é estabelecido e o contador de iterações (it) é inicializado em zero.

A cada iteração, o PPO é executado para prever e selecionar uma NSH com base no estado atual. A NSH escolhida é executada, gerando uma nova solução. Após a execução da NSH, o estado é atualizado. Se a nova solução for a melhor solução encontrada, a $melhor\_solução$ é atualizada. Caso contrário, uma perturbação é aplicada à solução atual para evitar que o algoritmo fique preso

em ótimos locais. Esse processo se repete até que o número de iterações atinja n (número de tarefas da instância) ou até que a função objetivo da solução seja igual ao Limite Inferior. A Figura 4.4 demonstra o esquema de execução do algoritmo PPO-VND.

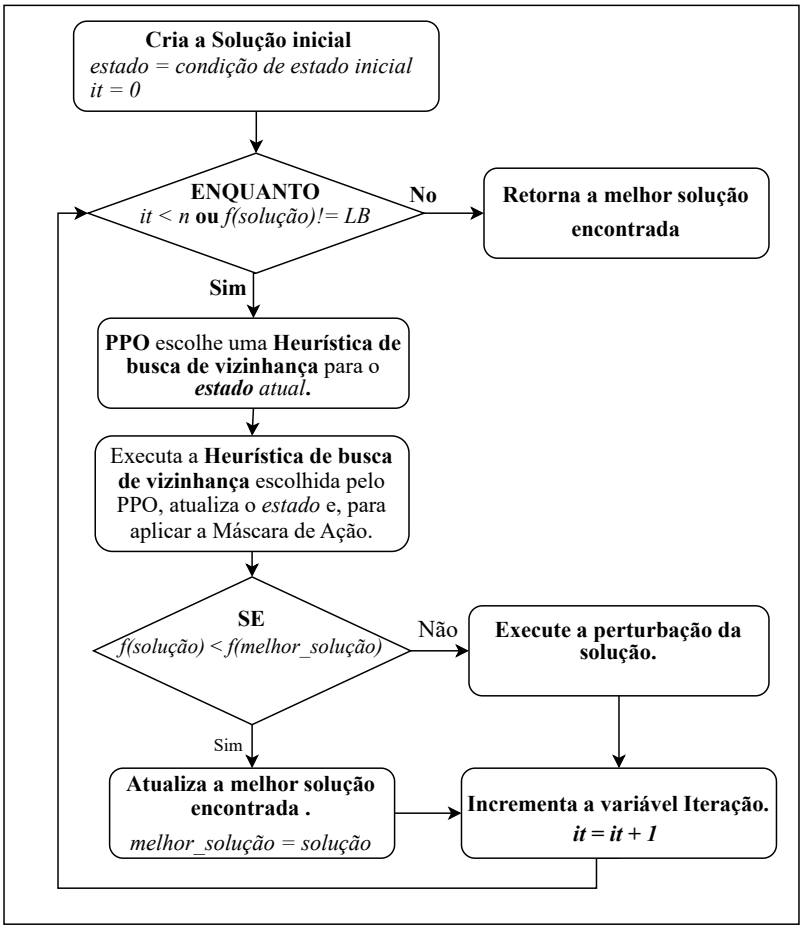


Figura 4.4. O esquema de execução do algoritmo PPO-VND.

Fonte: Próprio Autor

Algorithm 10 PPO - VND

```
1: solução ← SOLUÇÃOINICIAL()
2: melhor_solução ← solução
3: estado ← condição de estado inicial
4: it ← 0
5: enquanto it < n OU f(solução)! = LB faça
6:     u ← PPO.predict(estado) ▷ O PPO escolhe uma NSH para o estado atual.
7:     solução ← NSH u (solução)
8:     estado ← atualiza o estado
9:     se F(solução) < F(melhor_solução) então
10:        melhor_solução ← solução
11:    senão
12:        perturba a solução
13:    fim se
14:    it ← it + 1
15: fim enquanto
16: retorna melhor_solução
```

A meta-heurística híbrida PPO-VND integra de forma eficaz as vantagens do aprendizado por reforço com técnicas de busca local determinística na resolução de problemas de otimização combinatória. Ao aproveitar a capacidade preditiva do PPO para selecionar as heurísticas de busca de vizinhança mais adequada, o algoritmo consegue aprimorar tanto a eficiência quanto a qualidade das soluções obtidas.

Capítulo 5

Experimentos Computacionais

Este capítulo apresenta os métodos de criação das instâncias utilizadas nos experimentos computacionais e os resultados obtidos para avaliar o desempenho dos métodos propostos. Os Modelos de Programação Linear Inteira Mista foram implementados em C++ e resolvidos utilizando o *solver* CPLEX (Cplex, 2009), enquanto os demais métodos foram programados em *Python 3*. Todos os experimentos foram executados em um ambiente *single thread* utilizando uma CPU Intel Core i7-4790K (4,0 GHz) com 32 GB de memória RAM. Este capítulo detalha a configuração dos experimentos e a análise dos resultados, fornecendo uma visão abrangente da eficácia e eficiência dos métodos propostos.

5.1 Instâncias para o Problema

Para avaliar o desempenho dos algoritmos e a eficiência dos modelos matemáticos desenvolvidos, foi criado um algoritmo para gerar um conjunto de instâncias para o problema proposto. Essas instâncias também servem como base de comparação para futuros algoritmos.

Os valores dos parâmetros deste problema foram gerados com base em diversos estudos relevantes sobre problemas de programação de tarefas em máquinas e roteamento de veículos. Os tempos de processamento (p_{ij}) foram gerados aleatoriamente no intervalo [1,99], enquanto os tempos de preparação (s_{ijk}) foram gerados nos intervalos [1,9] ou [1,99], conforme o trabalho de Vallada & Ruiz (2011).

As localizações dos clientes foram geradas em um plano cartesiano ao redor do depósito, visando representar um cenário realista para o problema de roteamento de veículos. A distância euclidiana foi então utilizada para calcular o tempo de viagem entre o depósito e os clientes (t_{jk}), conforme a abordagem proposta por Li et al. (2007). Durante a geração dos clientes e o cálculo das distâncias, a propriedade da desigualdade triangular foi respeitada, garantindo que a distância direta entre dois pontos seja sempre menor ou igual à soma das distâncias ao passar por um terceiro ponto. Isso assegura a consistência e a viabilidade das rotas geradas, contribuindo para a robustez dos resultados obtidos.

As datas de entrega (d_j) das tarefas aos clientes são derivadas do trabalho de Lee et al. (2013). Os valores de d_j são definidos aleatoriamente no intervalo $[L(1-T-R/2), L(1+T+R/2)]$ e somados à média dos tempos de viagem ($\bar{t}$). Os parâmetros L, T e R representam, respectivamente, o *lower bound* do *makespan*, o fator de atraso e o intervalo das datas de entrega. O valor do *lower bound* do *makespan* (L) é calculado com base na Equação 5.1, onde p_j^{min} e s_j^{min} correspondem ao menor tempo de processamento e ao menor tempo de preparação da tarefa j. Os valores de T e R foram fixados em 0,2 e 0,8, respectivamente.

$$L = \frac{\sum\limits_{j \in I} (p_j^{min} + s_j^{min})}{m} \tag{5.1}$$

As demandas dos clientes (h_j) seguem o trabalho de Ullrich (2013b) e estão diretamente ligadas ao tempo de processamento da tarefa, de modo que tarefas com maiores tempos de processamento têm maior probabilidade de ocupar mais espaço no veículo. O valor de h_j é definido aleatoriamente no intervalo $[p_j^{min}, p_j^{médio}]$, onde $p_j^{médio}$ representa o tempo médio de processamento da tarefa j. As penalidades de atraso (w_j) das tarefas são geradas aleatoriamente no intervalo de [1,20].

A quantidade de veículos e suas capacidades são determinadas aleatoriamente, garantindo que a demanda total seja atendida com uma margem de 20%, 50% ou 90%. A capacidade de cada veículo é definida como $q_v = \max_{j=1....n}(h_j) + \tau_v$, onde τ_v é um valor aleatório no intervalo $[0, \mu * \max_{j=1....n}(h_j)]$, sendo μ um parâmetro de ajuste fixado em 2. Novos veículos são adicionados até que a capacidade total supere 120%, 150% ou 190% da demanda total dos clientes.

O Algoritmo 11 apresenta o processo de geração do grafo para o problema de roteamento de veículos. A Figura 5.1 ilustra o grafo gerado pelo Algoritmo 11. O Algoritmo 12 assegura a propriedade da desigualdade triangular no grafo gerado. Por fim, o Algoritmo 13 descreve o pseudocódigo para a geração de instâncias do problema proposto.

Os parâmetros utilizados para a geração das instâncias incluem: o número de máquinas ($m = \{2, 4, 8\}$), o número de tarefas ($n = \{10, 15, 30\}$), o tempo máximo de preparação ($setup = \{9, 99\}$), a distância média dos clientes até o depósito ($distância = \{20, 100\}$), e a capacidade total dos veículos ($Capacidade = \{1.2, 1.5, 1.9\}$) superior à demanda total dos clientes. Estes parâmetros foram combinados em um fatorial completo, resultando em todas as combinações possíveis. Para cada combinação, foram geradas 5 instâncias, totalizando 540 instâncias.

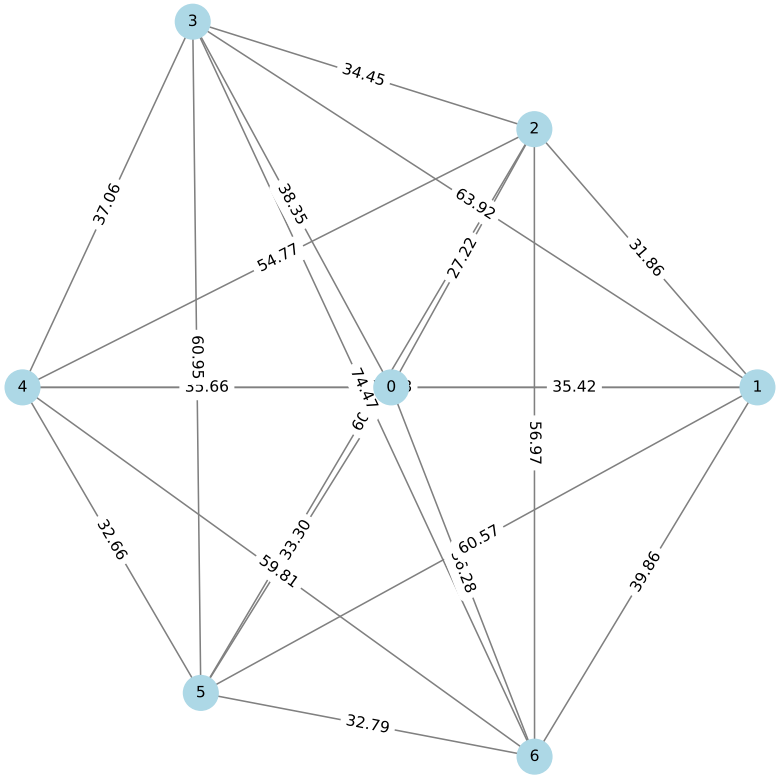


Figura 5.1. Disposição entre o depósito e os clientes

Fonte: Próprio Autor

Algorithm 11 Criação das localizações dos clientes.

função VRP_GERADOR($n, distancia$)
    $x[n+1]$
    $y[n+1]$
    $x[0] \leftarrow 0$
    $y[0] \leftarrow 0$
    para $j \leftarrow 1$ **ate** n **faça**
        $\gamma \leftarrow distancia * 1.1$
        $x[j] \leftarrow \gamma$ **cos** $[2(j-1)\pi/n]$ * **rand**$(0.75, 1.25)$
        $y[j] \leftarrow \gamma$ **sin** $[2(j-1)\pi/n]$ * **rand**$(0.75, 1.25)$
    fim para
    retorna x, y
fim função

Algorithm 12 Verificar Propriedade da Desigualdade Triangular no Grafo Gerado

```
1: função VerificarDesigualdadeTriângular(t)
2:     n ← dimensão de t
3:     para i ← 1 to n faça
4:         para j ← 1 to n faça
5:             para k ← 1 to n faça
6:                 se t[i][j] > t[i][k] + t[k][j] então
7:                     retorna False
8:                 fim se
9:             fim para
10:        fim para
11:    fim para
12:    retorna True
13: fim função
```

Algorithm 13 Algoritmo para geração de instâncias para o problema posposto.

1: **função** GERADOR_INSTANCIAS($m, n, setup, distancia, capacidade, T, R, \mu$)
2: $x, y \leftarrow$**VRP_Gerador**$(n, distancia)$
3: $\overline{t} \leftarrow 0$
4: **para** $j \leftarrow 0$ **ate** n **faça**
5: **para** $k \leftarrow 0$ **ate** n **faça**
6: $t[j][k] = \sqrt{(x[j] + x[k])^2 + (y[j] + y[k])^2}$
7: **se** $j \neq k$ **então**
8: $\overline{t} = \overline{t} + t[j][k]$
9: **fim se**
10: **fim para**
11: **fim para**
12: **se VerificarDesigualdadeTriângular** (t) **então**
13: Volte para a Linha 2
14: **fim se**
15: $\overline{t} = \overline{t}/(n * (n-1))$
16: **para** $i \leftarrow 0$ **ate** m **faça**
17: **para** $j \leftarrow 1$ **ate** n **faça**
18: $p[i][j] =$ **rand** (1,99)
19: **para** $k \leftarrow 1$ **ate** n **faça**
20: $s[i][j][k] =$ **rand** $(1, setup)$
21: **fim para**
22: **fim para**
23: **fim para**
24: $L = 0$
25: **para** $j \leftarrow 1$ **ate** n **faça**
26: $L = L + (p_j^{min} + s_j^{min})/m$
27: **fim para**
28: $h_{max} = 0$
29: $h_{total} = 0$
30: **para** $j \leftarrow 1$ **ate** n **faça**
31: $d[j] = \overline{t} +$ **rand** $(L(1 - T - R/2), L(1 - T + R/2))$
32: $w[j] =$ **rand** $(1, 20)$
33: $h[j] =$ **rand** (p_j^{min}, p_j^{media})
34: $h_{total} = h_{total} + h[j]$
35: **se** $h[j] > h_{max}$ **então**
36: $h_{max} = h[j]$
37: **fim se**
38: **fim para**
39: $h_{total} = h_{total} * capacidade$
40: **enquanto** $h_{total} > 0$ **faça**
41: $cap = h_{max} +$ **rand** $(1, (\mu * h_{max}))$
42: q. **push** (cap)
43: $h_{total} = h_{total} - cap$
44: **fim enquanto**
45: **fim função**

5.2 Resultados Computacionais

Neste capítulo, são apresentados os resultados dos experimentos computacionais realizados para avaliar o desempenho dos métodos propostos. Os algoritmos implementados neste trabalho estão disponíveis publicamente nos repositórios do perfil do *github* `https://github.com/matheusinhofreitas`.

Os métodos comparados neste capítulo incluem os modelos MILP resolvidos pelo CPLEX (Cplex, 2009), a meta-heurística RVND, a meta-heurística híbrida PPO-VND, o *Framework* e as oito NSH. Para cada algoritmo, foram analisadas a qualidade das soluções e a eficiência computacional.

Todos os métodos determinísticos foram executados uma única vez, enquanto os métodos não determinísticos, RVND e PPO-VND, foram executados 5 vezes. Nas comparações, os valores da função objetivo e do tempo de execução para as meta-heurísticas RVND e PPO-VND correspondem à média das cinco execuções dos algoritmos. O tempo de processamento para os modelos de Programação Linear Inteira Mista (MILP) foi fixado em 600 segundos. Nos experimentos com RVND e PPO-VND, o critério de parada foi definido pela quantidade de tarefas (n) da instância.

Para avaliar a qualidade das soluções obtidas pelos métodos propostos, foi calculado o Desvio Percentual Relativo (RPD, do inglês *Relative Percentage Deviation*) para cada uma das instâncias. O RPD, definido pela Equação 5.2, quantifica a diferença percentual entre a solução obtida e uma solução de referência, fornecendo uma medida clara e objetiva do desempenho do algoritmo em relação ao melhor valor conhecido ou à solução ótima.

$$RPD(\%) = 100 \times \frac{(F_{método} - F_{melhor})}{F_{método}} \tag{5.2}$$

onde F_{melhor} representa o valor da função objetivo da melhor solução conhecida para uma determinada instância do problema, obtida entre todos os métodos comparados, e $F_{método}$ representa o valor da função objetivo para o método em questão.

Adicionalmente, foi calculada a Melhoria Percentual Relativa (RPI, do inglês *Relative Percentage Improvement*) para todas as instâncias. Essa métrica é empregada para quantificar a porcentagem de melhoria alcançada por um algoritmo

em comparação com a solução inicial. O RPI, definido pela Equação 5.3, oferece uma medida precisa do ganho de desempenho proporcionado pelo algoritmo, permitindo avaliar de forma objetiva a eficácia da heurística aplicada.

$$RPI(\%) = 100 \times \frac{(F_{s.i.} - F_{método})}{F_{s.i.}} \tag{5.3}$$

onde $F_{s.i.}$ representa o valor da função objetivo da solução inicial fornecida pela heurística construtiva, e $F_{método}$ representa o valor da função objetivo para o método em questão.

Os resultados apresentados nesta seção fornecem uma visão detalhada sobre o desempenho relativo dos diferentes algoritmos, destacando as vantagens e limitações de cada abordagem no contexto do problema integrado de programação da produção e distribuição abordado neste trabalho.

5.2.1 Experimento 1: Comparações entre os Modelos MILP

Nesta subseção, comparamos os dois modelos de Programação Linear Inteira Mista (MILP 1 e MILP 2) propostos para resolver o problema de programação da produção e distribuição. Cada modelo possui características distintas que influenciam diretamente sua complexidade e desempenho computacional.

O MILP 1 baseia-se na representação de precedências e na roteirização, abordando de forma integrada a alocação de tarefas nas máquinas e a roteirização dos veículos. O modelo utiliza variáveis binárias para indicar a precedência das tarefas tanto nas máquinas quanto nas rotas dos veículos, além de definir a utilização dos veículos para rotas específicas. A complexidade do MILP 1 está principalmente associada ao número de variáveis binárias e às restrições que asseguram a correta ordem de execução das tarefas e a viabilidade das rotas. A natureza combinatória dessas variáveis de precedência adiciona um nível significativo de dificuldade ao problema, especialmente à medida que o número de tarefas, máquinas e veículos aumenta.

O MILP 2 adota uma abordagem indexada no tempo, baseada na discretização do horizonte temporal, em que as variáveis de decisão indicam os momentos exatos de início e término dos processos de cada tarefa, bem como suas respectivas entregas. Esse modelo oferece uma representação mais detalhada do problema,

proporcionando uma visão precisa dos períodos de execução das tarefas nas máquinas e suas entregas. No entanto, a discretização amplia significativamente o número de variáveis e restrições, resultando em um crescimento exponencial da complexidade computacional à medida que o horizonte temporal e o número de tarefas aumentam. Além disso, a necessidade de sincronizar o horizonte temporal do sequenciamento das tarefas nas máquinas com o das rotas dos veículos impõe um desafio adicional à resolução do modelo.

Na comparação entre os MILPs, o MILP 1 foi capaz de encontrar soluções válidas para todo o conjunto de 540 instâncias. Em contraste, o MILP 2 conseguiu resolver apenas 64 dessas instâncias. Das 64 instâncias resolvidas pelo MILP 2, 30 são de tamanho $n\_10\_m\_2$, 21 são de tamanho de $n\_10\_m\_4$, e 15 de tamanho $n\_10\_m\_8$, indicando que ele encontrou soluções apenas para instâncias menores, especialmente aquelas com 10 tarefas. A disparidade na quantidade de instâncias resolvidas por cada modelo evidencia a robustez do MILP 1, que se mostrou mais capaz de lidar com uma variedade maior de instâncias do problema.

Para uma comparação justa entre os modelos, utilizamos as 64 instâncias em que ambos conseguiram encontrar soluções válidas. O RPD do MILP 1 foi de 6%, enquanto o do MILP 2 alcançou 94%. Ademais, nenhuma das soluções encontradas pelo MILP 2 superou as obtidas pelo MILP 1. Esses resultados indicam que o MILP 1 não apenas se mostrou mais consistente na obtenção de soluções válidas, mas também produziu soluções de qualidade superior em comparação ao MILP 2, com RPDs significativamente menores.

Adicionalmente, realizamos uma Análise de Variância (ANOVA) (Zar, 1999) com um nível de significância de 5% para o grupo de 64 instâncias resolvidas. A ANOVA indicou um p-valor menor que 0,001, apontando diferenças estatisticamente significativas entre os MILP analisados. Esse resultado reforça a superioridade do MILP 1 em termos de desempenho e eficácia na resolução do problema proposto. As Figuras 5.2 e 5.3 ilustram os gráficos *boxplot*, bem como os intervalos e médias dos RPD obtidos pelos modelos matemáticos, evidenciando essa diferença.

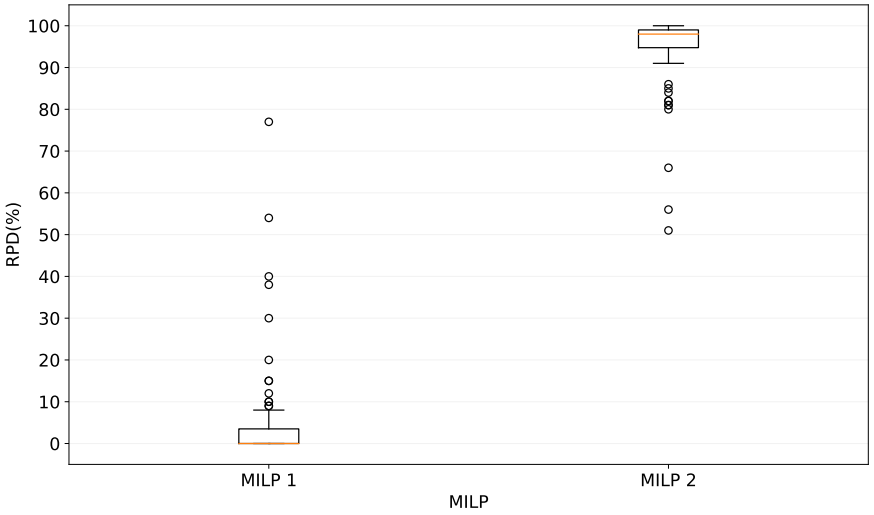


Figura 5.2. Gráfico *Boxplot* para o RPD (%) para os modelos MILP apresentados.

Fonte: Próprio Autor

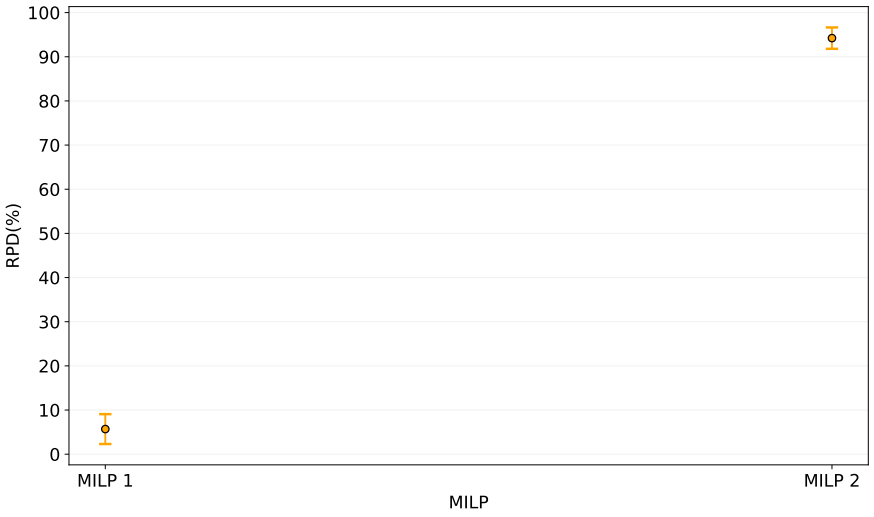


Figura 5.3. Gráfico de Intervalos e Médias para o RPD (%) para os modelos MILP apresentados.

Fonte: Próprio Autor

Em resumo, o MILP 1 demonstrou ser mais eficaz e eficiente, tanto na obtenção de soluções válidas para uma ampla gama de instâncias quanto na qualidade superior das soluções geradas. Em contrapartida, o MILP 2, apesar de fornecer

uma representação mais detalhada, enfrentou dificuldades significativas na resolução da maioria das instâncias, resultando em um desempenho geral inferior.

Devido ao baixo desempenho do MILP 2, evidenciado pela incapacidade de encontrar soluções válidas para a maioria das instâncias testadas e pela significativa diferença no RPD em relação ao MILP 1, decidimos utilizar apenas o MILP 1 nas próximas comparações. Essa escolha se baseia na necessidade de garantir uma análise robusta e confiável, utilizando o modelo comprovadamente mais eficaz e eficiente para a resolução do problema integrado de programação da produção e distribuição.

5.2.2 Experimento 2: Comparações entre o Framework e as Heurísticas de Busca de Vizinhança

Nesta subseção, apresentamos os resultados do experimento que compara o desempenho do *Framework* proposto com as Heurísticas de Busca de Vizinhança desenvolvidas. O *Framework*, que utiliza técnicas de aprendizado de máquina, foi projetado para prever e selecionar a NSH mais eficaz para resolver cada instância do problema. Essa abordagem permite que o *Framework* adapte dinamicamente sua estratégia de solução conforme as características de cada instância, melhorando potencialmente a eficácia e eficiência do processo de otimização. Durante a execução do *Framework* foram identificados 15 grupos distintos (*clusters*) de instâncias com base em suas características estatísticas. O treinamento da rede neural durou cerca de 4 horas. Entretanto, uma vez treinado, o modelo pode recomendar a NSH ideal para qualquer instância em aproximadamente 0,1 segundos, tornando o *Framework* extremamente eficiente para aplicações práticas.

Por outro lado, as NSH implementadas utilizam estratégias específicas de troca (*swap*) e inserção (*insertion*) para explorar o espaço de busca, movendo-se iterativamente entre soluções vizinhas. Essas heurísticas são métodos clássicos e bem-estabelecidos em problemas de otimização combinatória. Elas oferecem um caminho direto e eficiente para aprimorar as soluções iniciais, garantindo uma melhoria contínua e consistente no processo de otimização.

Os valores de RPD são apresentados na Tabela 5.1. Cada linha da tabela representa os valores médios de RPD das 60 instâncias, classificados com base

no número de tarefas e máquinas. Os menores RPD médios são destacados em negrito.

Tabela 5.1. RPD (%) para o *Framework* e as Heurísticas de Busca de Vizinhança.

Instâncias	*Framework*	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	**49,1%**	82,8%	81,0%	84,8%	82,5%	77,5%	**49,1%**	79,4%	65,7%
N_10_M_4	**49,1%**	72,7%	69,3%	76,4%	72,6%	64,5%	49,4%	69,5%	56,4%
N_10_M_8	**29,4%**	45,6%	40,8%	53,8%	47,1%	41,1%	**29,4%**	44,2%	34,4%
N_15_M_2	**35,9%**	88,0%	85,4%	89,4%	87,3%	81,3%	38,2%	85,2%	59,9%
N_15_M_4	**38,8%**	71,3%	64,0%	74,6%	69,4%	65,0%	39,7%	69,2%	49,8%
N_15_M_8	**27,9%**	51,7%	38,8%	57,8%	48,1%	46,9%	29,7%	53,9%	35,8%
N_30_M_2	**31,4%**	90,4%	86,5%	91,4%	89,6%	89,2%	33,1%	89,7%	73,9%
N_30_M_4	**32,4%**	75,0%	64,3%	77,0%	72,5%	69,7%	33,0%	72,0%	56,5%
N_30_M_8	30,4%	59,9%	44,2%	62,8%	57,4%	53,8%	**29,7%**	57,6%	42,7%
Média	**36,1%**	70,8%	63,8%	74,2%	69,6%	65,4%	36,8%	69,0%	52,8%

Fonte: Próprio Autor

Os menores valores de RPD médios, que indicam um melhor desempenho, estão destacados em negrito para facilitar a comparação. As Figuras 5.4 e 5.5 apresentam, respectivamente, o *boxplot* e o gráfico de intervalos e médias dos RPD, fornecendo uma visualização clara da distribuição dos valores de RPD para o *Framework* e as Heurísticas de Busca de Vizinhança (NSH).

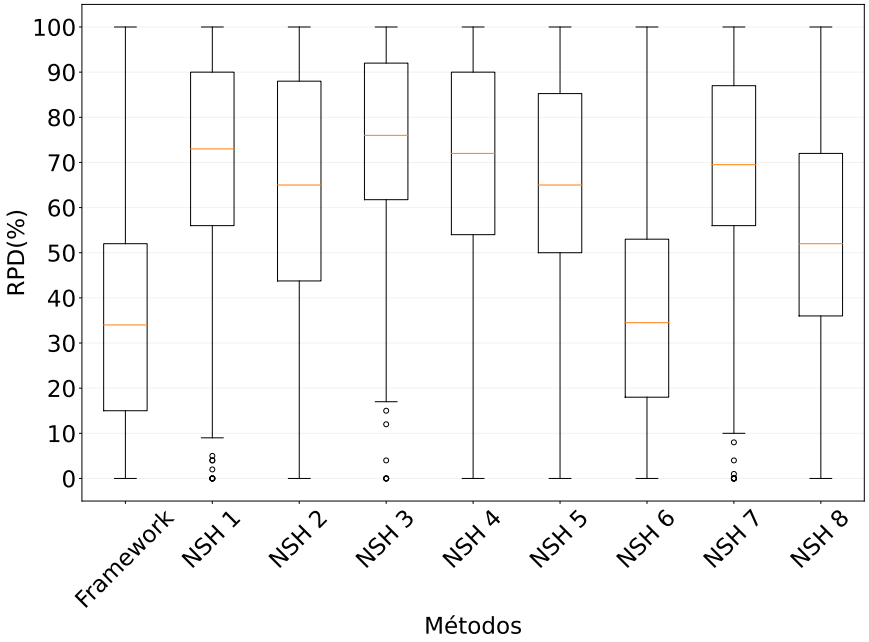


Figura 5.4. Gráfico *Boxplot* para o RPD (%) para o *Framework* e as Heurísticas de Busca de Vizinhança.

Fonte: Próprio Autor

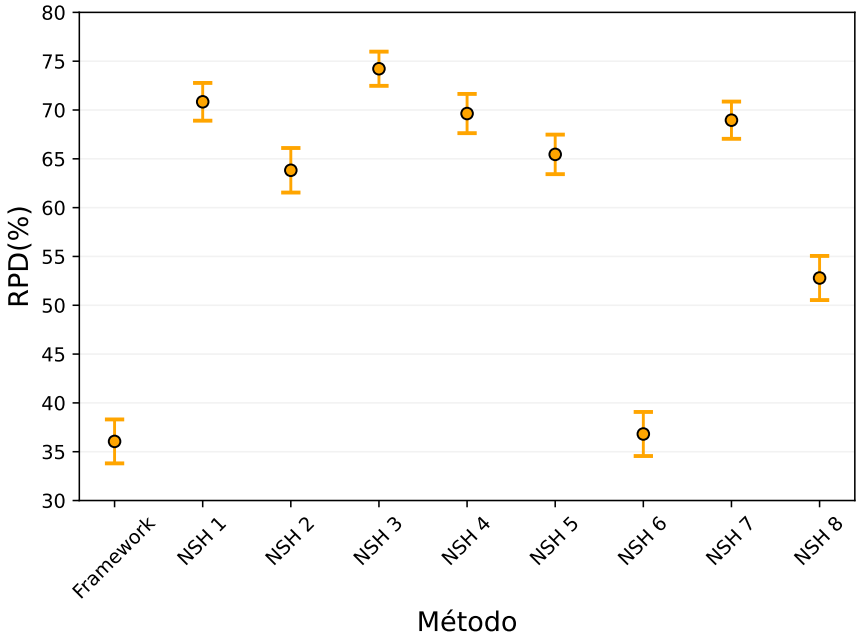


Figura 5.5. Gráfico de Intervalos e Médias o RPD (%) para o *Framework* e as Heurísticas de Busca de Vizinhança.

Fonte: Próprio Autor

Inicialmente, analisamos o desempenho das NSH utilizando o RPD como a principal métrica para avaliar a qualidade das soluções encontradas por cada uma. Observou-se que a NSH 6 apresentou o melhor desempenho entre as heurísticas, alcançando um RPD médio de 36,8%. Em contrapartida, as demais heurísticas apresentaram RPD médios mais altos, refletindo uma menor eficácia na obtenção de soluções de alta qualidade. Por exemplo, a NSH 1 obteve um RPD médio de 70,8%, a NSH 2 registrou 63,8%, e a NSH 3 alcançou 74,2%. Essas diferenças significativas nos RPD médios ressaltam a variabilidade na eficácia das diferentes heurísticas de busca de vizinhança, destacando a superioridade da NSH 6 em comparação com as demais.

A NSH 6 se destaca por aplicar movimentos de troca na lista de tarefas $L1$ e reiniciar o processo sempre que uma solução melhor que a inicial é encontrada. Ao comparar apenas as heurísticas, a NSH 6 demonstrou uma clara superioridade, encontrando os melhores valores para 386 (71%) instâncias. A NSH 8 obteve 139

(26%) instâncias, enquanto a NSH 2 conseguiu os melhores valores em 89 (16%) instâncias.

As demais heurísticas apresentaram desempenhos inferiores: a NSH 5 encontrou os melhores valores para 47 (9%) instâncias, a NSH 4 para 41 (8%), a NSH 7 para 37 (7%), a NSH 1 para 32 (6%), e a NSH 3 para 26 (5%) instâncias, respectivamente. Esses resultados reforçam a eficácia da NSH 6 e sua capacidade de superar consistentemente as demais heurísticas em um número significativo de instâncias.

Quando comparado ao *Framework*, que integra técnicas de aprendizado de máquina e NSH, o desempenho das heurísticas individuais mostrou-se inferior. O *Framework* destacou-se ao encontrar os melhores RPD em 8 dos 9 grupos de instâncias, alcançando um RPD médio de 36,1%, um resultado ligeiramente melhor que o da NSH 6, que obteve 36,8%. Esses resultados evidenciam a eficácia do *Framework* em adaptar-se às características das instâncias e superar as heurísticas individuais na otimização do problema proposto.

A Tabela 5.2 apresenta os resultados obtidos pelos algoritmos em relação ao RPI. Este índice mede a eficácia das heurísticas em proporcionar melhorias em relação à solução inicial, com valores mais altos indicando maior eficiência. Os valores do RPI das NSH e do *Framework* revelam diferenças significativas no desempenho de cada método.

Ao analisar os resultados das NSH individualmente, observamos que a NSH 6 e a NSH 8 se destacaram em diversos grupos de instâncias. Por exemplo, para a instância de tamanho N_10_M_2, a NSH 6 atinge um RPI de 75,8%, enquanto a NSH 8 alcança 66,7%. Em contrapartida, a NSH 1 e a NSH 3 apresentaram desempenho inferior na maioria das instâncias, com RPI médio de 27,7% e 18,3%, respectivamente. Esses resultados evidenciam que algumas heurísticas possuem maior potencial para melhorar a solução inicial em determinados grupos de instâncias.

Ao comparar o *Framework* com as NSH, observamos que o *Framework* supera consistentemente todas as heurísticas, exceto a NSH 6. Nas instâncias analisadas, a NSH 6 apresentou os melhores RPI na maioria dos grupos de instâncias, com um RPI médio de 67,5%, ligeiramente superior ao do *Framework* (67,0%). As Figuras 5.6 e 5.7 apresentam, respectivamente, o *boxplot* e o gráfico de intervalos e médias

do RPI, fornecendo uma visualização clara da distribuição dos valores de RPI para o *Framework* e as Heurísticas de Busca de Vizinhança.

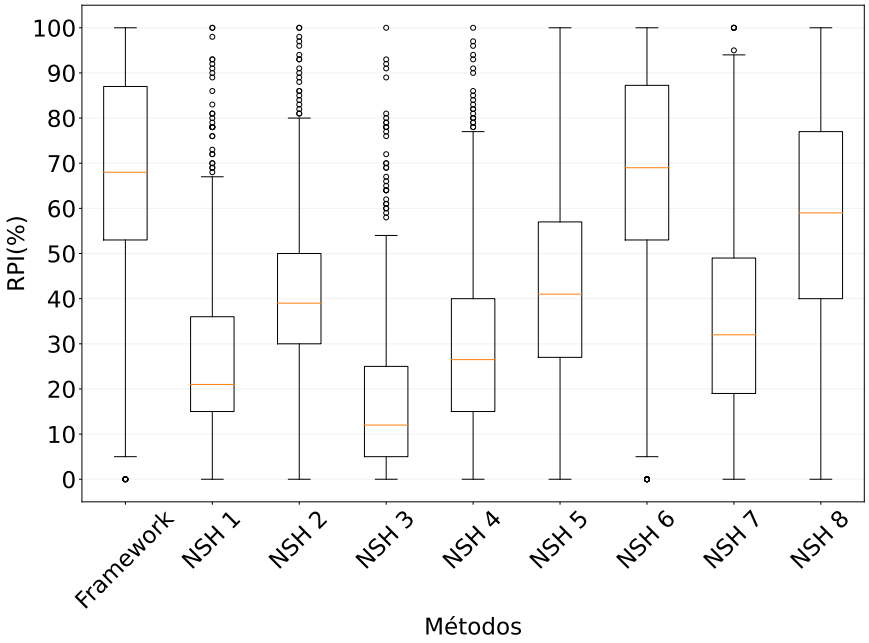


Figura 5.6. Gráfico *Boxplot* para o RPI (%) para o *Framework* e as Heurísticas de Busca de Vizinhança.

Fonte: Próprio Autor

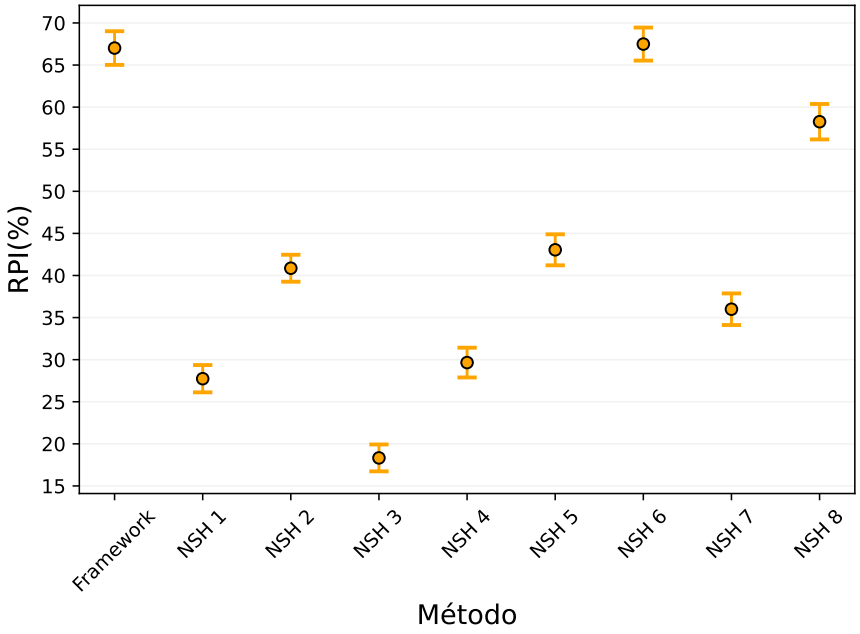


Figura 5.7. Gráfico de Intervalos e Médias do RPI (%) para o *Framework* e as Heurísticas de Busca de Vizinhança.

Fonte: Próprio Autor

Após uma análise mais profunda dos resultados encontrados pela NSH 6 e pelo *Framework*, realizamos um teste ANOVA (Zar, 1999) com um nível de significância de 5% para comparar os resultados obtidos pelos dois algoritmos. A ANOVA resultou em um p-valor de 0,64, indicando que não existem diferenças estatisticamente significativas entre o *Framework* e a NSH 6.

O resultado do teste ANOVA comparando o *Framework* com a NSH 6 é consistente com o cenário em que um algoritmo se destaca significativamente para resolver a maioria das instâncias. Esse é o caso da NSH 6. O *Framework* é treinado para selecionar a melhor NSH para cada instância do problema. Como a NSH 6 foi a mais eficaz para a maioria das instâncias, o *Framework* optou por utilizá-la em grande parte dos casos.

A eficácia do *Framework* em comparação às heurísticas é evidenciada por sua alta taxa de assertividade na escolha do algoritmo mais adequado para cada instância. Neste experimento, o *Framework* demonstrou uma assertividade de 72%, selecionando a melhor heurística para 391 instâncias. Além disso, o *Framework* foi

responsável por encontrar 73 soluções que determinaram o valor F_{melhor} no cálculo do RPD, enquanto a NSH 6 encontrou 64 soluções. Essa ligeira superioridade do *Framework* pode ser atribuída à sua capacidade de ajustar dinamicamente a NSH mais apropriada com base nas características específicas de cada instância do problema.

Os resultados apresentados nesta subseção destacam a superioridade do *Framework* em comparação com as Heurísticas de Busca de Vizinhança individuais. O *Framework* mostrou uma capacidade aprimorada de adaptação e otimização, alcançando um desempenho superior na maioria das instâncias avaliadas. A abordagem híbrida, que combina técnicas de aprendizado de máquina com heurísticas de busca local, permitiu ao *Framework* selecionar dinamicamente a heurística mais adequada para cada instância, maximizando a qualidade das soluções obtidas.

A NSH 6 destacou-se como a melhor heurística individual, apresentando RPD e RPI altamente competitivos em relação ao *Framework*. No entanto, a capacidade do *Framework* de ajustar sua estratégia com base nas características específicas de cada instância proporcionou uma leve vantagem em termos de assertividade e qualidade das soluções finais. Este experimento reafirma a importância de utilizar abordagens híbridas e adaptativas em problemas de otimização combinatória, demonstrando que a integração de diferentes técnicas pode trazer melhorias significativas no desempenho e na eficiência dos algoritmos.

Tabela 5.2. RPI (%) para o *Framework* e as Heurísticas de Busca de Vizinhança.

Instâncias	*Framework*	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	**75,8%**	31,0%	36,4%	21,3%	30,0%	53,3%	**75,8%**	47,5%	66,7%
N_10_M_4	63,3%	35,4%	42,3%	25,2%	34,9%	51,2%	**65,3%**	42,7%	58,3%
N_10_M_8	51,3%	39,5%	44,6%	28,2%	36,9%	44,8%	**52,7%**	40,4%	49,2%
N_15_M_2	83,4%	27,1%	38,0%	17,9%	29,5%	54,6%	**84,0%**	46,0%	74,8%
N_15_M_4	64,8%	26,7%	39,5%	17,1%	28,6%	42,7%	**64,9%**	33,2%	57,5%
N_15_M_8	**55,8%**	36,1%	48,7%	25,6%	38,0%	42,6%	55,6%	35,1%	51,2%
N_30_M_2	**87,0%**	16,6%	36,0%	7,9%	21,5%	31,9%	**87,0%**	25,9%	70,0%
N_30_M_4	**68,2%**	16,7%	39,6%	8,9%	22,8%	35,4%	67,8%	28,8%	53,0%
N_30_M_8	53,5%	20,2%	42,7%	12,9%	24,7%	30,9%	**54,3%**	24,0%	43,8%
Média	67,0%	27,7%	40,9%	18,3%	29,7%	43,0%	**67,5%**	36,0%	58,3%

Fonte: Próprio Autor

5.2.3 Experimento 3: Treinamento do Algoritmo PPO

Nesta seção, são apresentados os resultados obtidos pelo algoritmo PPO durante o processo de aprendizado para resolver o problema integrado de programação da produção e distribuição. O treinamento do algoritmo PPO partiu de soluções iniciais, e as curvas de aprendizado mostram um aumento consistente na recompensa média ao longo do tempo. Esses resultados evidenciam a eficácia do processo de aprendizado, indicando que o algoritmo é capaz de melhorar suas decisões progressivamente, adaptando-se de forma eficiente às complexidades do problema.

As figuras nesta seção apresentam as curvas de aprendizado dos agentes PPO para diferentes tamanhos de instâncias: $N = 10$, $N = 15$ e $N = 30$. O eixo vertical indica o total de recompensas acumuladas por episódio, enquanto o eixo horizontal representa o número de períodos ao longo do treinamento.

As curvas mais claras nos gráficos refletem os valores das recompensas ao longo dos episódios. Cada ponto nessa curva corresponde à recompensa total obtida pelo agente em um único episódio. Essa representação permite monitorar diretamente a variabilidade e a evolução das recompensas durante o treinamento, evidenciando como o desempenho do agente responde às iterações realizadas.

Por outro lado, as curvas mais escuras apresentam uma versão suavizada da curva de recompensas, utilizando o método de suavização (*smoothing*) com um parâmetro ($\alpha = 1$). Essa suavização é implementada por meio de uma média móvel exponencial, que reduz a influência de flutuações abruptas nos valores e enfatiza a tendência geral de aprendizado do agente do PPO ao longo dos episódios. Essa abordagem facilita a identificação de padrões e oferece *insights* mais claros sobre o progresso do treinamento, como a presença de melhorias consistentes ou a estabilidade no desempenho do agente à medida que o treinamento avança.

A Figura 5.8, destacada em laranja, mostra a curva de aprendizado para as instâncias de tamanho $N = 10$. Nota-se que o algoritmo PPO atinge um aprendizado mais rápido em instâncias menores, estabilizando a recompensa média já nos primeiros 100 mil episódios. Esse resultado sugere que, para problemas de menor complexidade, o algoritmo PPO é capaz de aprender de maneira eficiente em um período relativamente curto.

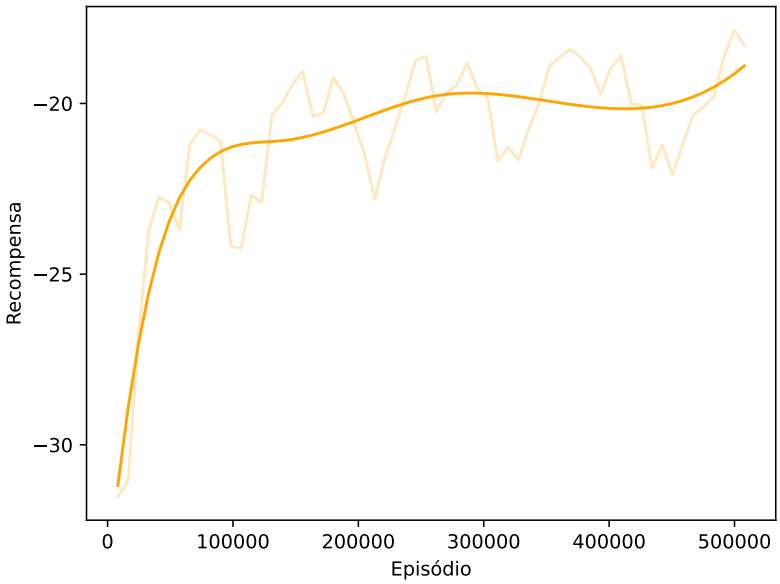


Figura 5.8. Curva de Aprendizado do PPO para instâncias de tamanho N = 10

Fonte: Próprio Autor

Nas instâncias de tamanho $N = 15$ e $N = 30$, representadas pelos gráficos azul e verde nas Figuras 5.9 e 5.10, respectivamente, o PPO enfrenta maiores dificuldades no processo de aprendizado. A maior complexidade dessas instâncias demanda um número significativamente maior de episódios para que as recompensas se estabilizem. Esse comportamento é esperado, uma vez que problemas mais complexos exigem um tempo de treinamento mais prolongado para que o algoritmo possa aprender de forma eficaz.

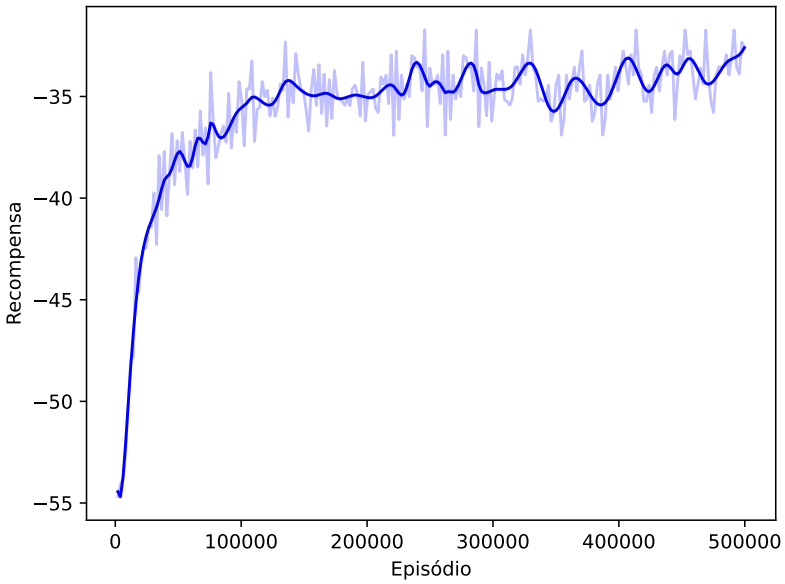


Figura 5.9. Curva de Aprendizado do PPO para instâncias de tamanho N = 15

Fonte: Próprio Autor

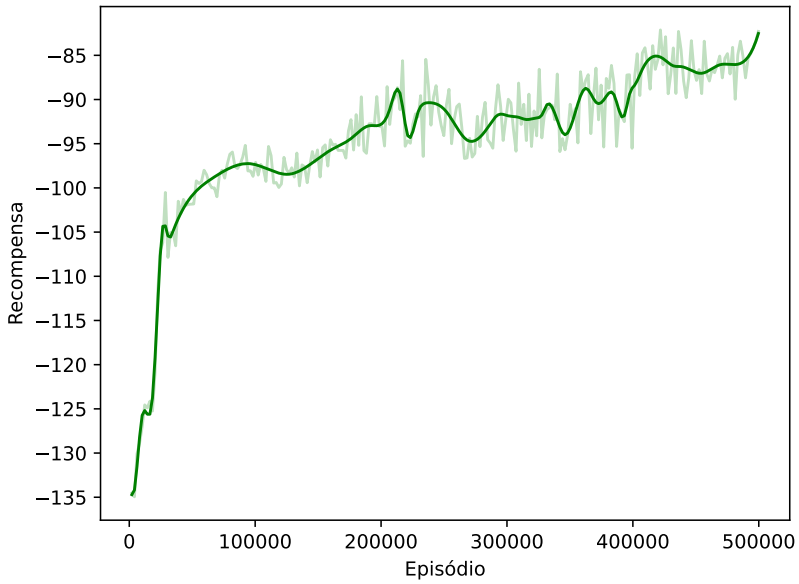


Figura 5.10. Curva de Aprendizado do PPO para instâncias de tamanho N = 30

Fonte: Próprio Autor

O tempo de treinamento do PPO variou conforme o tamanho das instâncias: cerca de 16 horas para $N = 10$, aproximadamente 74 horas para $N = 15$ e em torno de 45 dias para $N = 30$. Vale destacar que o tempo de treinamento do modelo PPO não foi incluído nas comparações diretas, pois o treinamento é realizado apenas uma vez, enquanto a execução do modelo treinado pode ser repetida várias vezes com eficiência.

A análise das curvas de aprendizado e do desempenho do algoritmo PPO demonstra sua capacidade de adaptação e melhoria contínua. O PPO aprende a partir de soluções iniciais, otimizando o processo de programação da produção e distribuição, mesmo em cenários de alta complexidade.

Em conclusão, o algoritmo PPO demonstrou uma capacidade robusta de aprendizado, especialmente em instâncias menores, mostrando-se eficaz na resolução de problemas complexos quando combinado com outras técnicas de otimização. A habilidade do PPO de se adaptar e aprimorar continuamente com o treinamento é um ponto forte que justifica seu uso em problemas de otimização combinatória e inteligência artificial.

5.2.4 Experimento 4: Calibração do Algoritmo RVND

Para determinar o valor ideal para os parâmetros do algoritmo RVND, foram realizados testes computacionais preliminares para calibrar a taxa de perturbação, utilizada para modificar as listas de tarefas L1 e L2 que geram a solução durante a execução do RVND. Foram testados os seguintes valores para a taxa de perturbação: $\{10\%, 20\%, 30\%, 40\%, 50\%\}$ do valor de n, resultando em cinco experimentos distintos.

Para realizar os testes, foi utilizado um subconjunto de 63 instâncias escolhidas aleatoriamente, sendo 21 de cada tamanho de tarefas. Os resultados obtidos foram avaliados usando a média do RPD calculado para cada uma das instâncias, com cada instância sendo executada cinco vezes. Os resultados também foram analisados por meio do teste ANOVA.

As três principais hipóteses da ANOVA (normalidade, igualdade de variância e independência dos resíduos) foram verificadas. A Figura 5.11 apresenta o gráfico de intervalos de confiança HSD de *Tukey*, com nível de confiança de 95%, utilizado para comparar o desempenho de cada taxa de perturbação. A análise do teste revela que não há diferenças significativas entre os diferentes valores do parâmetro. No entanto, o RVND obteve melhores médias com a taxa de perturbação de 30%, razão pela qual esse valor foi selecionado para uso tanto no RVND quanto no PPO-VND.

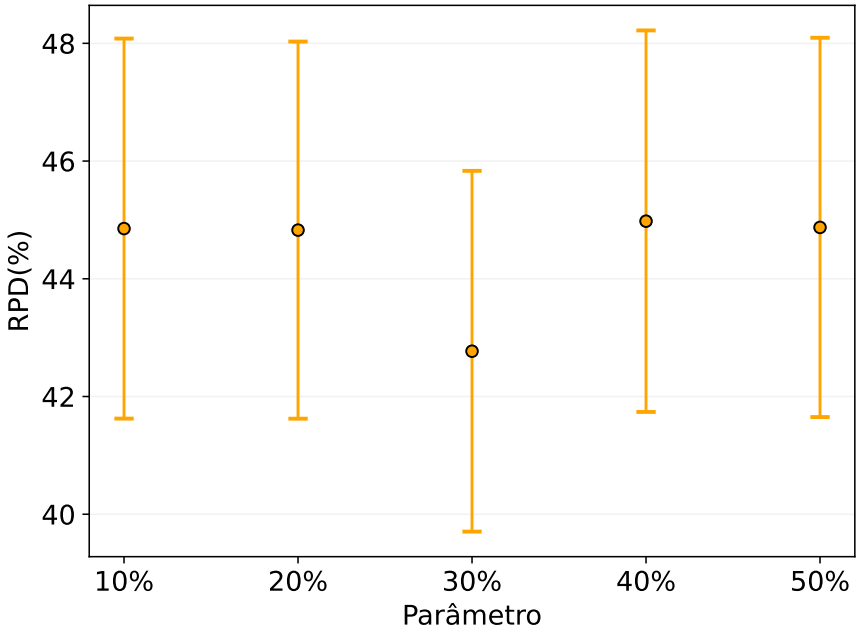


Figura 5.11. Gráfico de Intervalos e Médias RPD para Calibração do Parâmetro Pertubação da solução.

Fonte: Próprio Autor

5.2.5 Experimento 5: Comparações entre o MILP, Framework, RVND e PPO-VND

Nesta subseção, comparamos quatro metodologias distintas para resolver o problema integrado de programação da produção e roteamento de veículos. As metodologias analisadas são o MILP 1, o *Framework*, a meta-heurística RVND e a abordagem híbrida PPO-VND. O objetivo é avaliar a eficácia, eficiência e aplicabilidade dessas metodologias, utilizando os valores de RPD e Índice de Melhoria Percentual Relativa (RPI), conforme mostrado nas Tabelas 5.3 e 5.4. As Figuras 5.12 e 5.13 apresentam os gráficos *boxplot* para o RPD e RPI, respectivamente, referentes aos métodos analisados.

O MILP é eficaz na obtenção de soluções ótimas para problemas com restrições complexas, mas torna-se impraticável para instâncias de grande escala devido ao tempo de computação exponencial necessário. A meta-heurística RVND é eficiente em encontrar soluções rapidamente; entretanto, pode ficar presa em ótimos

locais e sua eficácia depende da qualidade da solução inicial. A abordagem híbrida PPO-VND combina aprendizado por reforço, que ajusta políticas de decisão em tempo real, com a exploração de diversas vizinhanças pelo VND. Essa abordagem é adaptativa, mas requer um grande volume de dados e um tempo considerável para treinamento.

Para verificar se existem diferenças significativas entre os métodos analisados, foi realizada o teste ANOVA (Zar, 1999) com um nível de confiança de 5%. A ANOVA resultou em um p-valor de 0,00, indicando que há diferenças estatisticamente significativas entre os métodos analisados. A Figura 5.14 apresenta o gráfico de intervalos e médias de RPD para os métodos analisados nesta subseção.

Nas instâncias com 10 tarefas, o MILP 1 destacou-se com os menores RPDs em todas as configurações de máquinas (N_10_M_2, N_10_M_4 e N_10_M_8), demonstrando sua eficácia em cenários de menor complexidade. Embora inferior ao MILP 1, o PPO-VND superou o RVND e o *Framework* nas instâncias N_10_M_2 e N_10_M_4. Por outro lado, o RVND teve melhor desempenho em N_10_M_8, com um RPD de 9,7% em sua melhor execução e um RPD médio de 20,6%. Além disso, durante a execução do MILP 1, o modelo conseguiu encontrar 34 soluções ótimas, sendo 25 delas nas instâncias com 10 tarefas.

Com o aumento para 15 tarefas, o PPO-VND destacou-se nas instâncias N_15_M_2, mostrando eficácia no gerenciamento de complexidade moderada. O RVND apresentou os melhores resultados em N_15_M_8, tanto na média quanto nas melhores execuções. O MILP 1 teve um desempenho notável em N_15_M_4, embora o RVND e o PPO-VND tenham obtido melhores resultados em suas melhores execuções.

Para as instâncias com 30 tarefas, a complexidade aumentada evidenciou a robustez do PPO-VND, que consistentemente obteve os melhores valores de RPD em todas as configurações de máquinas (N_30_M_2, N_30_M_4 e N_30_M_8), com RPD de 26,3%, 23,6% e 18,6%, respectivamente. Em comparação, o RVND e o *Framework* apresentaram desempenhos inferiores, confirmando a superioridade do PPO-VND em cenários de maior complexidade.

Essa diferença significativa destaca a capacidade adaptativa do PPO-VND em ajustar suas políticas de busca local para gerenciar eficientemente um número maior de tarefas. Em contraste, o MILP 1 apresentou os maiores valores de RPD,

evidenciando uma redução significativa em seu desempenho quando aplicado a instâncias maiores.

Para o conjunto completo de instâncias, o PPO-VND apresentou o menor RPD médio (28,7%), seguido pelo *Framework* (36,1%), MILP 1 (40,1%) e RVND (41,1%). Embora o RVND não tenha apresentado um RPD médio tão favorável, ele superou as outras abordagens em suas melhores execuções, exceto o PPO-VND. Esses resultados indicam que, enquanto o MILP 1 é eficaz em instâncias menores, o PPO-VND e o RVND são mais adequados para cenários complexos e de maior escala.

Além dos valores de RPD, analisamos o RPI para medir a eficácia das heurísticas em melhorar a solução inicial. Os resultados comprovam que praticamente todos os métodos avaliados (PPO-VND, RVND e *Framework*) apresentaram uma melhora significativa em relação à solução inicial.

Nas instâncias com 10 tarefas, o PPO-VND destacou-se com os maiores RPI, alcançando 79,7% nas instâncias de configuração N_10_M_2 e 70% nas de configuração N_10_M_4. Em contraste, o RVND obteve o maior RPI para as instâncias da configuração N_10_M_8 (58,8%).

Conforme o número de tarefas aumentou para 15, observou-se um padrão semelhante, com o PPO-VND obtendo os melhores resultados para as instâncias das configurações N_15_M_2 e N_15_M_4, com RPI de 84,6% e 70,8%, respectivamente. Por outro lado, o RVND foi especialmente eficaz nas instâncias de tamanho N_15_M_8, alcançando o melhor RPI de 61,7%.

Nas instâncias com 30 tarefas, o PPO-VND demonstrou eficiência ao lidar com instâncias de alta complexidade. Esse algoritmo apresentou um desempenho superior em todas as configurações de 30 tarefas, demonstrando sua adaptabilidade e robustez em cenários de maior demanda.

No conjunto total de instâncias, o *Framework* apresentou o menor RPI médio, com 67,0%, seguido pelo RVND com 68,6% e o PPO-VND com 71,4%. O PPO-VND destacou-se por aprimorar significativamente as soluções iniciais, especialmente em instâncias mais complexas.

Este estudo comparativo revela a importância de selecionar a metodologia de otimização conforme a complexidade específica do problema. O MILP 1 mostrou-se altamente eficaz em cenários de menor complexidade, ao passo que o PPO-

VND destacou-se em situações de alta complexidade, graças à sua escalabilidade e adaptabilidade.

Esses achados oferecem diretrizes valiosas para a escolha de metodologias de otimização em problemas reais. Eles sugerem que futuras pesquisas explorem ajustes nas parametrizações do RVND e do *Framework*, além de investigar a integração de técnicas de aprendizado de máquina em meta-heurísticas para problemas de otimização combinatória.

Tabela 5.3. RPD (%) para o MILP, Framework, RVND e PPO-VND.

Instâncias	RPD				Melhor RPD		Pior RPD	
	PPO-VND	RVND	Framework	MILP 1	PPO-VND	RVND	PO-VND	RVND
N_10_M_2	41,1%	59,0%	49,1%	**7,2%**	34,1%	45,1%	45,8%	69,7%
N_10_M_4	42,7%	45,0%	49,1%	**4,9%**	35,5%	32,0%	48,6%	57,0%
N_10_M_8	22,4%	20,6%	29,4%	**2,8%**	16,0%	9,7%	27,5%	33,2%
N_15_M_2	**33,6%**	65,1%	35,9%	37,6%	21,6%	50,9%	39,7%	75,4%
N_15_M_4	24,2%	35,5%	38,8%	**21,3%**	14,3%	16,7%	32,1%	51,1%
N_15_M_8	25,7%	**20,0%**	27,9%	37,2%	12,9%	4,7%	37,4%	32,5%
N_30_M_2	**26,3%**	69,1%	31,4%	79,5%	4,4%	56,0%	43,2%	77,4%
N_30_M_4	**23,6%**	34,4%	32,4%	80,5%	9,9%	15,4%	35,4%	46,8%
N_30_M_8	**18,6%**	21,5%	30,4%	89,5%	7,0%	8,7%	26,4%	31,0%
Média	**28,7%**	41,1%	36,1%	40,1%	17,3%	26,6%	37,3%	52,7%

Fonte: Próprio Autor

Tabela 5.4. RPI (%) para o Framework, RVND e PPO-VND.

Instâncias	RPI			Pior RPI		Melhor RPI	
	PPO-VND	RVND	Framework	PPO-VND	RVND	PPO-VND	RVND
N_10_M_2	**79,7%**	73,7%	75,8%	77,4%	65,2%	82,8%	80,9%
N_10_M_4	**70,0%**	68,7%	63,3%	66,2%	60,3%	74,1%	75,4%
N_10_M_8	57,2%	**58,8%**	51,3%	54,0%	52,7%	60,7%	63,6%
N_15_M_2	**84,6%**	77,9%	83,4%	82,9%	70,4%	87,2%	84,8%
N_15_M_4	**70,8%**	67,7%	64,8%	67,5%	60,3%	74,9%	75,0%
N_15_M_8	58,7%	**61,7%**	55,8%	51,9%	55,3%	65,2%	67,6%
N_30_M_2	**88,5%**	80,0%	87,0%	84,9%	74,2%	91,7%	85,0%
N_30_M_4	**72,9%**	69,8%	68,2%	68,3%	64,5%	77,3%	75,5%
N_30_M_8	**60,5%**	59,2%	53,5%	56,9%	54,4%	64,9%	64,5%
Média	**71,4%**	68,6%	67,0%	67,8%	61,9%	75,4%	74,7%

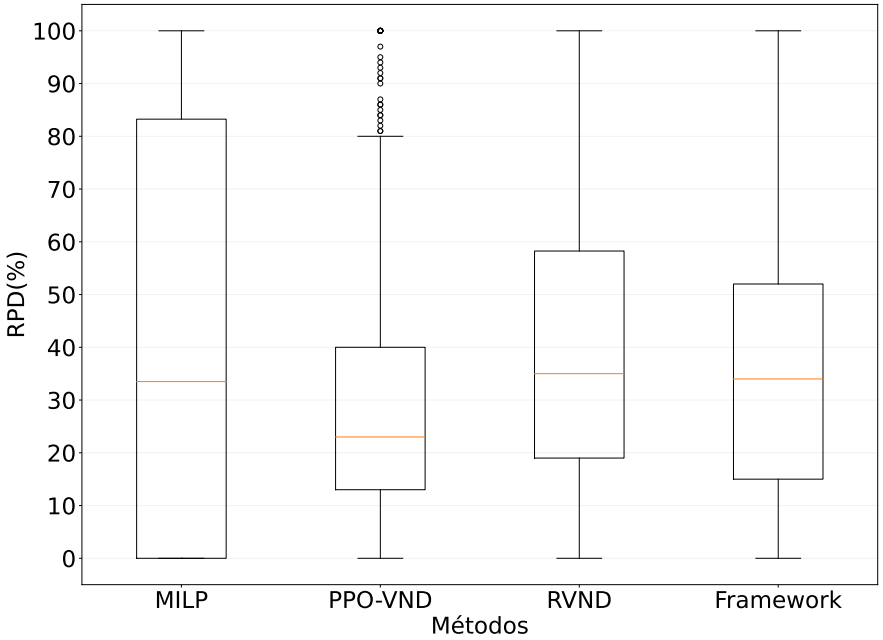


Figura 5.12. Gráfico *Boxplot* RPD para o MILP, PPO-VND, RVND e *Framework*.

Fonte: Próprio Autor

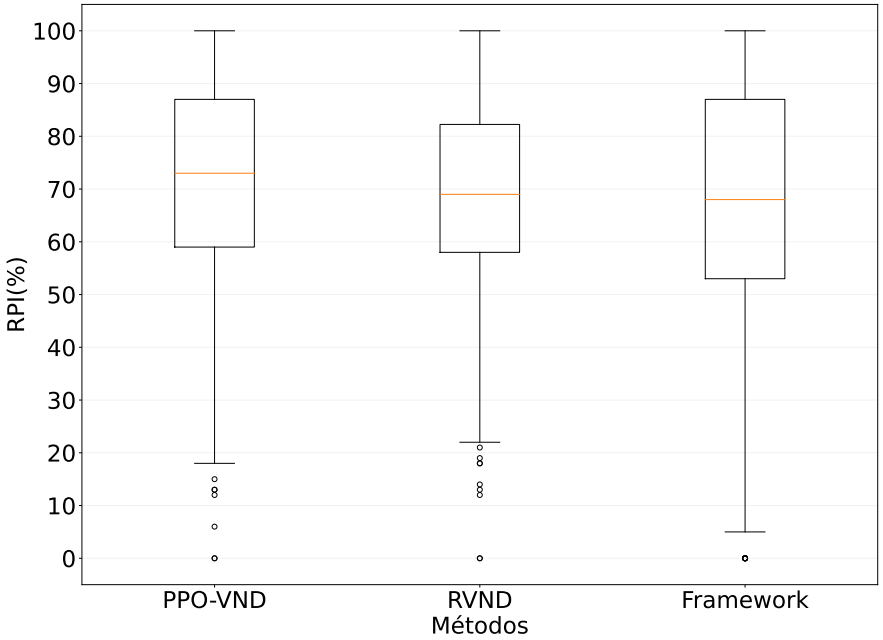


Figura 5.13. Gráfico *Boxplot* RPI para o PPO-VND, RVND e *Framework*.

Fonte: Próprio Autor

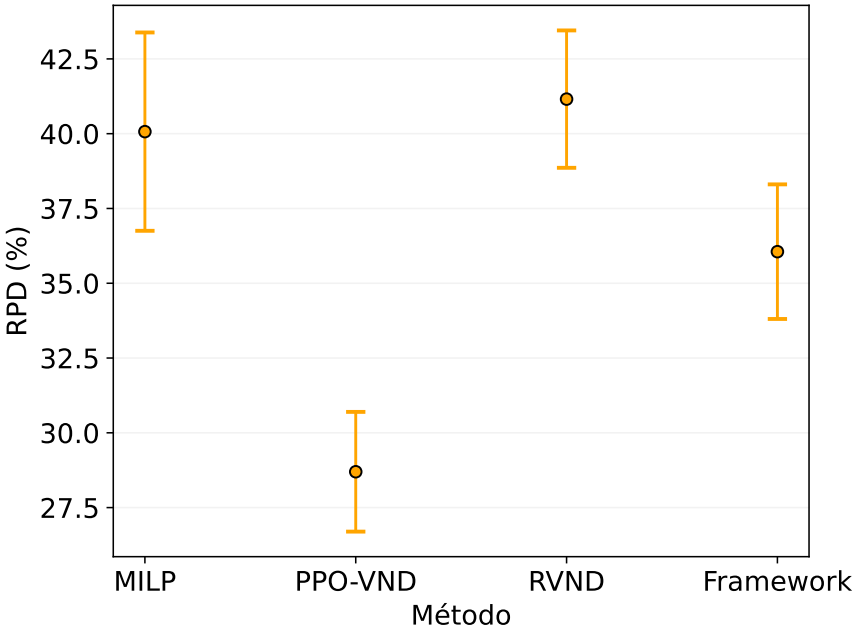


Figura 5.14. Gráfico de Intervalos e Médias RPD para o MILP, *Framework*, RVND e PPO-VND.

Fonte: Próprio Autor

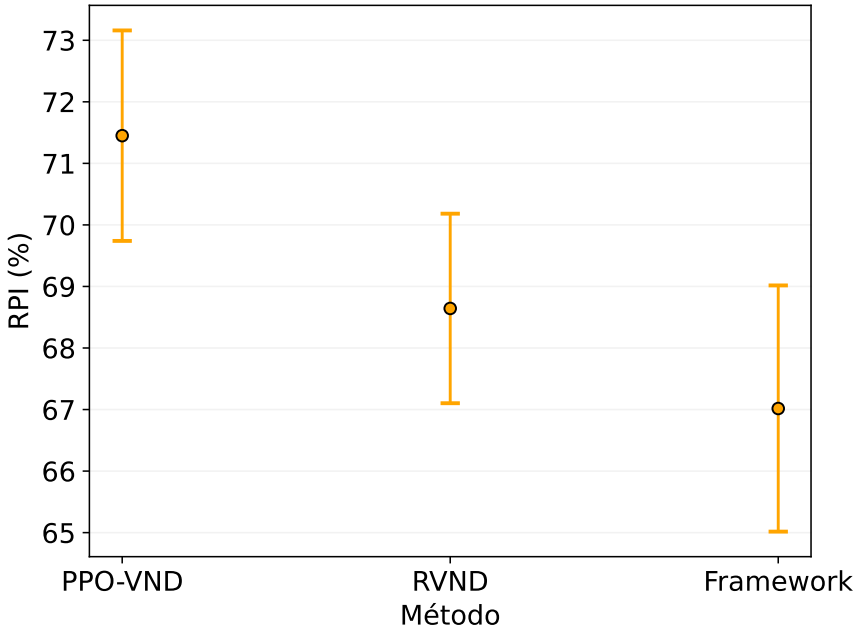


Figura 5.15. Gráfico de Intervalos e Médias RPI para o *Framework*, RVND e PPO-VND.

Fonte: Próprio Autor

5.2.6 Experimento 6: Visão Geral dos Resultados

Neste capítulo, apresentamos uma visão geral dos resultados obtidos na comparação das metodologias de Programação Linear Inteira Mista (MILP 1), *Framework*, *Random Variable Neighborhood Descent* (RVND), *Proximal Policy Optimization* com *Variable Neighborhood Descent* (PPO-VND) e as heurísticas de busca local (NSH).

Nesta seção, sintetizamos as comparações realizadas nos capítulos anteriores, enfocando os tempos de processamento, o Desvio Percentual Relativo (RPD), o Índice de Melhoria Percentual Relativa (RPI), além da análise visual por meio de gráficos de intervalos, médias e *boxplots*.

A Tabela 5.5 apresenta os tempos de processamento médios para cada método, classificados pelo número de tarefas e máquinas. Observamos que o MILP 1, embora eficaz em instâncias menores, apresenta tempos de processamento significativamente maiores. Por outro lado, o PPO-VND demonstrou tempos de

processamento mais equilibrados e eficientes em todas as configurações testadas. O RVND, embora eficaz na obtenção de boas soluções, possui um tempo de processamento duas vezes maior que o PPO-VND para as instâncias maiores ($n = 30$). O *Framework* e as NSH, por sua vez, apresentam tempos de processamento significativamente menores que as meta-heurísticas, uma vez que esses algoritmos são executados apenas uma vez, enquanto as meta-heurísticas executam os NSH no mínimo n vezes. Além disso, as NSH que possuem *restart* apresentam um tempo de processamento maior em comparação às que não possuem essa característica.

Tabela 5.5. Tempo de execução do algoritmo em segundos.

Intancias	MILP	PPO-VND	RVND	Framework	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	564,70	1,46	1,53	0,57	0,16	0,27	0,30	0,72	0,15	0,29	0,28	0,72
N_10_M_4	547,06	1,21	1,85	0,44	0,19	0,30	0,37	0,84	0,18	0,35	0,34	0,87
N_10_M_8	528,21	2,02	2,38	0,56	0,26	0,39	0,51	1,03	0,26	0,45	0,48	1,23
N_15_M_2	593,68	7,40	11,61	0,90	0,71	1,06	1,38	3,91	0,70	1,24	1,28	3,56
N_15_M_4	599,71	10,12	13,76	1,04	0,83	1,38	1,62	4,83	0,82	1,56	1,53	4,51
N_15_M_8	598,07	17,61	17,68	1,54	1,13	1,83	2,23	5,06	1,13	2,05	2,09	6,23
N_30_M_2	600,57	146,83	300,57	12,96	10,62	16,55	20,61	82,44	10,54	18,23	19,85	63,48
N_30_M_4	600,68	167,22	354,30	15,78	12,29	19,96	24,20	100,94	12,38	21,55	23,28	69,80
N_30_M_8	601,20	269,30	469,47	19,75	16,20	24,62	32,17	113,66	17,78	27,54	30,66	88,10
Média	581,54	69,24	130,35	5,95	4,71	7,37	9,27	34,83	4,88	8,14	8,87	26,50

Fonte: Próprio Autor

A Tabela 5.6 apresenta os valores de RPD para cada método. Esses valores são fundamentais para entender a qualidade das soluções encontradas por cada metodologia em diferentes cenários.

O MILP se mostrou extremamente eficiente para resolver as instâncias pequenas ($n = 10$); entretanto, à medida que a complexidade das instâncias aumenta, o MILP não consegue produzir resultados de alta qualidade. Mesmo assim, nesses experimentos, o MILP se destacou como o quarto melhor método para resolver o problema, com um RPD médio de 40,1% para todo o conjunto de instâncias.

O método com melhor RPD médio é o PPO-VND, com um RPD médio de 28,7%. Além disso, essa meta-heurística se mostrou extremamente eficiente para instâncias de grande porte ($n = 30$). O *Framework* demonstrou ser um método com grande potencial para resolver esse problema, além de ser um dos métodos mais eficientes em termos de tempo de processamento. Esse método conseguiu encontrar boas soluções para a maioria das instâncias, obtendo o segundo melhor RPD médio (36,1%), seguido pela NSH 6, com um RPD médio de 36,8%.

Neste caso, podemos observar que tanto o *Framework* quanto o NSH 6 melhoram a qualidade das soluções à medida que o número de tarefas aumenta.

A meta-heurística RVND, por sua vez, alcançou um RPD médio de 41,1%, um valor ligeiramente melhor que o encontrado pelo MILP, sendo superior ao MILP para resolver instâncias maiores. Além disso, o RVND superou todas as NSH, exceto o NSH 6.

As NSH podem ser classificadas da seguinte forma: NSH 6 com um RPD médio de 36,7%, seguida por NSH 8 (52,8%), NSH 2 (63,8%), NSH 5 (65,4%), NSH 7 (69,0%), NSH 4 (69,6%), NSH 1 (70,8%) e NSH 3 (74,2%).

Para facilitar a compreensão dos resultados, as Figuras 5.16 e 5.17 apresentam *boxplots* e gráficos de intervalos e médias do RPD.

Os gráficos de médias apresentam a tendência geral de desempenho dos métodos, enquanto os *boxplots* fornecem uma visão detalhada da distribuição dos resultados, destacando a variabilidade e a presença de *outliers*.

Tabela 5.6. RPD (%) para todos os métodos

Instances	RPD											
	MILP	PPO-VND	RVND	Framework	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	**7,2%**	41,1%	59,0%	49,1%	82,8%	81,0%	84,8%	82,5%	77,5%	49,1%	79,4%	65,7%
N_10_M_4	**4,9%**	42,7%	45,0%	49,1%	72,7%	69,3%	76,4%	72,6%	64,5%	49,4%	69,5%	56,4%
N_10_M_8	**2,8%**	22,4%	20,6%	29,4%	45,6%	40,8%	53,8%	47,1%	41,1%	29,4%	44,2%	34,4%
N_15_M_2	37,6%	**33,6%**	65,1%	35,9%	88,0%	85,4%	89,4%	87,3%	81,3%	38,2%	85,2%	59,9%
N_15_M_4	**21,3%**	24,2%	35,5%	38,8%	71,3%	64,0%	74,6%	69,4%	65,0%	39,7%	69,2%	49,8%
N_15_M_8	37,2%	25,7%	**20,0%**	27,9%	51,7%	38,8%	57,8%	48,1%	46,9%	29,7%	53,9%	35,8%
N_30_M_2	79,5%	**26,3%**	69,1%	31,4%	90,4%	86,5%	91,4%	89,6%	89,2%	33,1%	89,7%	73,9%
N_30_M_4	80,5%	**23,6%**	34,4%	32,4%	75,0%	64,3%	77,0%	72,5%	69,7%	33,0%	72,0%	56,5%
N_30_M_8	89,5%	**18,6%**	21,5%	30,4%	59,9%	44,2%	62,8%	57,4%	53,8%	29,7%	57,6%	42,7%
Média	40,1%	**28,7%**	41,1%	36,1%	70,8%	63,8%	74,2%	69,6%	65,4%	36,8%	69,0%	52,8%

Fonte: Próprio Autor

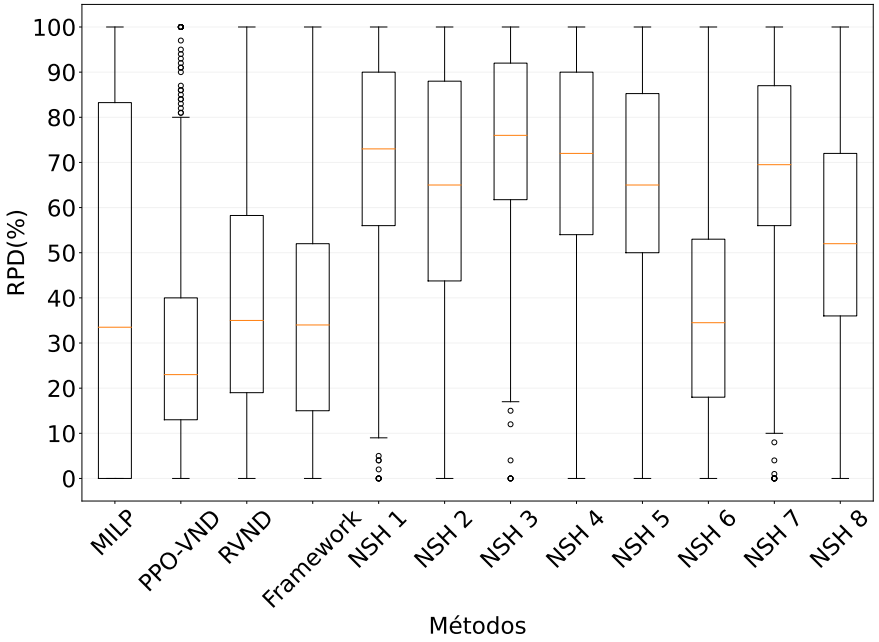


Figura 5.16. Gráfico *Boxplot* RPD para todos os métodos.

Fonte: Próprio Autor

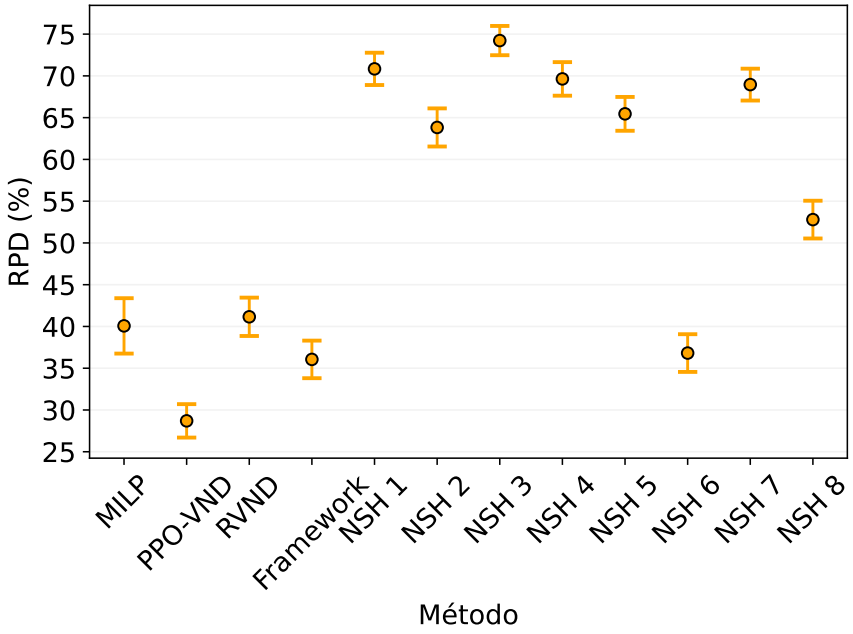


Figura 5.17. Gráfico de Intervalos e Médias RPD para todos os métodos.

Fonte: Próprio Autor

A Tabela 5.7 apresenta os resultados de RPI para cada método, sendo fundamentais para compreender a melhoria proporcionada por cada metodologia em diferentes instâncias.

O método com melhor RPI médio é o PPO-VND, com 71,4%. Além disso, essa meta-heurística mostrou-se extremamente eficiente para a maioria das instâncias.

A meta-heurística RVND apresentou o segundo melhor desempenho, com um RPI médio de 68,6%. Em terceiro e quarto lugares, o NSH 6 e o *Framework* obtiveram RPI médios de 67,5% e 67,0%, respectivamente.

As demais heurísticas podem ser classificadas da seguinte forma: NSH 8, NSH 5, NSH 2, NSH 7, NSH 4, NSH 1 e NSH 3, com RPI médios de 58,3%, 43,0%, 40,9%, 36,0%, 29,7%, 27,7% e 18,3%, respectivamente.

Para facilitar a compreensão dos resultados, as Figuras 5.18 e 5.19 apresentam *boxplots* e gráficos de médias dos RPI.

Os gráficos de médias apresentam a tendência geral de desempenho dos métodos, enquanto os *boxplots* fornecem uma visão detalhada da distribuição dos resultados, destacando a variabilidade e a presença de *outliers*.

Tabela 5.7. RPI (%) para todos os métodos

Instances	RPI										
	PPO-VND	RVND	Framework	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	**79,7%**	73,7%	75,8%	31,0%	36,4%	21,3%	30,0%	53,3%	75,8%	47,5%	66,7%
N_10_M_4	**70,0%**	68,7%	63,3%	35,4%	42,3%	25,2%	34,9%	51,2%	65,3%	42,7%	58,3%
N_10_M_8	57,2%	**58,8%**	51,3%	39,5%	44,6%	28,2%	36,9%	44,8%	52,7%	40,4%	49,1%
N_15_M_2	**84,6%**	77,9%	83,4%	27,1%	38,0%	17,9%	29,5%	54,6%	84,0%	46,0%	74,8%
N_15_M_4	**70,8%**	67,7%	64,8%	26,7%	39,5%	17,1%	28,6%	42,7%	64,9%	33,2%	57,5%
N_15_M_8	58,7%	**61,7%**	55,8%	36,1%	48,7%	25,6%	38,0%	42,6%	55,6%	35,1%	51,2%
N_30_M_2	**88,5%**	80,0%	87,0%	16,6%	36,0%	7,9%	21,5%	31,9%	87,0%	25,9%	70,0%
N_30_M_4	**72,9%**	69,8%	68,2%	16,7%	39,6%	8,9%	22,8%	35,4%	67,8%	28,8%	53,0%
N_30_M_8	**60,5%**	59,2%	53,5%	20,2%	42,7%	12,9%	24,7%	30,8%	54,3%	24,0%	43,8%
Média	**71,4%**	68,6%	67,0%	27,7%	40,9%	18,3%	29,7%	43,0%	67,5%	36,0%	58,3%

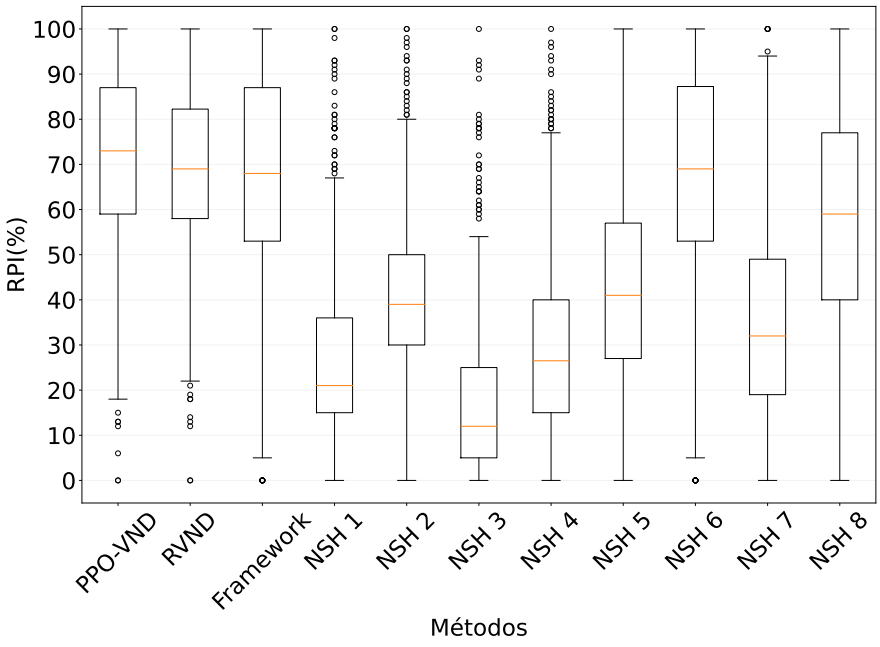


Figura 5.18. Gráfico *Boxplot* RPI para todos os métodos.

Fonte: Próprio Autor

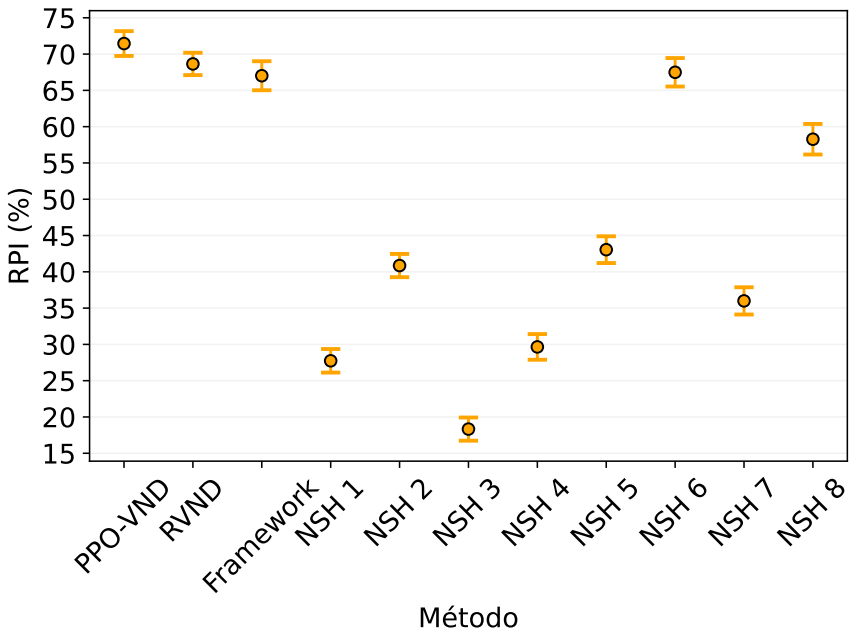


Figura 5.19. Gráfico de Intervalos e Médias RPI para todos os métodos.

Fonte: Próprio Autor

A análise comparativa detalhada dos métodos desenvolvidos e testados ao longo desta tese revelou as particularidades e as vantagens específicas de cada abordagem, levando em consideração a complexidade e o tamanho das instâncias. Em termos gerais, os principais pontos observados ao longo das comparações são:

- **MILP 1:** O modelo de Programação Linear Inteira Mista demonstrou ser altamente eficaz em instâncias menores, caracterizadas por um espaço de busca limitado. O MILP 1 conseguiu encontrar soluções ótimas para algumas instâncias pequenas ($n = 10$), destacando-se em termos de precisão e qualidade das soluções. No entanto, à medida que a complexidade das instâncias aumentou, especialmente para $n = 15$ e $n = 30$, o desempenho do MILP 1 reduziu-se significativamente, não conseguindo encontrar soluções satisfatórias dentro do tempo limite em cenários mais complexos.

- **NSH:** As heurísticas de busca de vizinhança apresentaram desempenho variado, dependendo da configuração do algoritmo. Entre as heurísticas testa-

das, a NSH 6 se destacou como a mais eficaz em alguns grupos de instâncias, conseguindo melhorar significativamente a qualidade das soluções iniciais.

- ***Framework*:** O *Framework* baseado em aprendizado de máquina demonstrou um grande potencial ao selecionar dinamicamente as heurísticas mais adequadas para resolver cada instância. Este método conseguiu equilibrar qualidade da solução e tempo de processamento, destacando-se como uma abordagem promissora. O *Framework* mostrou-se adaptativo e eficiente, superando várias heurísticas individuais em termos de desempenho global.

- **RVND:** A meta-heurística *Random Variable Neighborhood Descent* obteve bons resultados em instâncias de maior porte, superando o MILP 1 tanto em tempo de processamento quanto na qualidade das soluções. No entanto, o RVND foi consistentemente superado pelo PPO-VND e pelo *Framework*, evidenciando a vantagem de métodos que incorporam elementos de aprendizado adaptativo na exploração do espaço de soluções.

- **PPO-VND:** O algoritmo híbrido *Proximal Policy Optimization* com *Variable Neighborhood Descent* destacou-se como a abordagem mais eficaz e robusta entre as testadas. Esse método obteve consistentemente os menores valores de Desvio Percentual Relativo e os maiores Índices de Melhoria Percentual Relativa, especialmente em instâncias maiores ($n = 15$ e $n = 30$). A combinação do aprendizado por reforço com a meta-heurística permitiu ao PPO-VND ajustar dinamicamente suas políticas de busca, demonstrando uma clara superioridade em cenários de alta complexidade.

Em resumo, cada método analisado apresenta suas vantagens e desvantagens. O MILP 1 mostrou-se mais adequado para instâncias menores, enquanto o PPO-VND provou ser a solução mais escalável e adaptativa para instâncias de maior complexidade. O *Framework* baseado em aprendizado de máquina também se destaca como uma abordagem promissora para desenvolvimentos futuros, combinando flexibilidade e eficiência para lidar com a diversidade de cenários encontrados na prática.

Capítulo 6

Conclusões

Este trabalho abordou a complexidade inerente aos problemas integrados de programação da produção e roteamento de veículos, propondo e avaliando uma gama de metodologias que combinam técnicas tradicionais de otimização com abordagens avançadas de *inteligência artificial*. A integração desses dois domínios mostrou-se essencial para a resolução eficiente de problemas de alta complexidade, comuns em cenários industriais e logísticos modernos.

Os resultados obtidos ao longo desta pesquisa confirmam a eficácia do algoritmo híbrido PPO-VND, que combinou o poder das técnicas de aprendizado por reforço com a eficiência das meta-heurísticas de busca local. O PPO-VND destacou-se em termos de desempenho, especialmente em instâncias maiores, superando não apenas o modelo de MILP, mas também outras heurísticas e meta-heurísticas propostas. Esse desempenho superior reflete a capacidade adaptativa e escalável do PPO-VND, que mostrou-se robusto ao lidar com o aumento da complexidade dos problemas.

Além disso, a implementação do *Framework* baseado em aprendizado de máquina para a seleção de heurísticas de busca de vizinhança foi outro ponto de destaque nesta pesquisa. O *Framework* apresentou resultados competitivos, especialmente em comparação com as heurísticas tradicionais, demonstrando que a aplicação de técnicas de aprendizado de máquina na seleção e combinação de algoritmos pode gerar melhorias significativas na qualidade das soluções obtidas.

Apesar da eficácia comprovada do MILP em instâncias menores, sua limitação em lidar com problemas de maior escala evidenciou a necessidade de metodologias que combinem precisão com eficiência computacional. Nesse sentido, as meta-heurísticas, e em particular o PPO-VND, provaram ser ferramentas indispensáveis para a solução de problemas de otimização combinatória em ambientes complexos e dinâmicos.

As contribuições desta tese não se limitam à proposta de novos algoritmos, mas também incluem a criação de um conjunto de instâncias de teste, a definição de métricas claras de avaliação de desempenho e a comparação sistemática de metodologias diversas. Esses elementos formam uma base sólida para futuras pesquisas, sugerindo caminhos promissores para a continuidade do estudo, como a integração de técnicas de aprendizado de máquina, o desenvolvimento de algoritmos híbridos e a exploração de novos domínios de aplicação.

Em síntese, esta tese demonstra que a combinação de inteligência artificial e métodos de otimização combinatória não apenas potencializa a resolução de problemas complexos, mas também abre novas fronteiras para a pesquisa em ciência da computação aplicada à indústria. O trabalho realizado aqui estabelece um marco importante e contribui de forma significativa para o avanço na área de otimização combinatória e inteligência artificial.

Referências Bibliográficas

Allahverdi, A.; Gupta, J. N. & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2):219--239.

Allahverdi, A.; Ng, C. T.; Cheng, T. E. & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European journal of operational research*, 187(3):985--1032.

Alves, J. C. & Mateus, G. R. (2020). Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. Em *International Conference on Computational Logistics*, pp. 584--599. Springer.

Avalos-Rosales, O.; Angel-Bello, F. & Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 76:1705--1718.

Baker, K. R. & Merten, A. G. (1973). Scheduling with parallel processors and linear delay costs. *Naval Research Logistics Quarterly*, 20(4):793--804.

Boudia, M.; Louly, M. A. O. & Prins, C. (2008). Fast heuristics for a combined production planning and vehicle routing problem. *Production Planning and Control*, 19(2):85--96.

Chen, Z.-L. (2010). Integrated production and outbound distribution scheduling: review and extensions. *Operations research*, 58(1):130--148.

Cheng, T. & Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271--292.

Cheng, T. E.; Gupta, J. N. & Wang, G. (2000). A review of flowshop scheduling research with setup times. *Production and operations management*, 9(3):262--282.

Coffman, Jr., E. (1976). Computer and job shop scheduling theory.

Conway, R. W. (1967). "theory of scheduling,". *Addison Wesley*.

Cordeau, J.-F.; Laporte, G.; Savelsbergh, M. W. & Vigo, D. (2007). Vehicle routing. *Handbooks in operations research and management science*, 14:367--428.

Cplex, I. I. (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157.

Dantzig, G. B. & Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80--91.

Duan, J.-H.; Meng, T.; Chen, Q.-D. & Pan, Q.-K. (2018). An effective artificial bee colony for distributed lot-streaming flowshop scheduling problem. pp. 795--806.

Felix, G. P. & Arroyo, J. E. C. (2020). HeurÍsticas para o sequenciamento da produÇÃo e roteamento de veÍculos com frota heterogÊnea. *LII Simpósio Brasileiro de Pesquisa Operacional*.

Félix, G. P.; Arroyo, J. E. C. & de Freitas, M. (2023). Iterated local search heuristic for integrated single machine scheduling and vehicle routing. Em Kumar, S.; Sharma, H.; Balachandran, K.; Kim, J. H. & Bansal, J. C., editores, *Third Congress on Intelligent Systems*, pp. 223--235, Singapore. Springer Nature Singapore.

Fleszar, K.; Charalambous, C. & Hindi, K. S. (2012). A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 23(5):1949--1958.

Geismar, H. N.; Laporte, G.; Lei, L. & Sriskandarajah, C. (2008). The integrated production and transportation scheduling problem for a product with a short lifespan. *INFORMS Journal on Computing*, 20(1):21--33.

Ghinato, P. (1995). Sistema toyota de produção: mais do que simplesmente just-in-time. *Production*, 5:169--189.

Golden, B. L.; Raghavan, S. & Wasil, E. A. (2008). *The vehicle routing problem: latest advances and new challenges*, volume 43. Springer Science & Business Media.

Hansen, P. & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449--467.

Hou, Y.; Fu, Y.; Gao, K.; Zhang, H. & Sadollah, A. (2022). Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows. *Expert Systems with Applications*, 187:115827.

Karaoğlan, İ. & Kesen, S. E. (2017). The coordinated production and transportation scheduling problem with a time-sensitive product: a branch-and-cut algorithm. *International Journal of Production Research*, 55(2):536--557.

Kayhan, B. M. & Yildiz, G. (2021). Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. *Journal of Intelligent Manufacturing*, pp. 1--25.

Kim, D.-W.; Kim, K.-H.; Jang, W. & Chen, F. F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18(3-4):223--231.

Kim, D.-W.; Na, D.-G. & Chen, F. F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, 19(1-2):173--181.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kumar, S. N. & Panneerselvam, R. (2012). A survey on the vehicle routing problem and its variants.

Kurdi, M. (2020). A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem. *Applied Soft Computing*, 94:106458.

Laporte, G. & Nobert, Y. (1987). Exact algorithms for the vehicle routing problem. Em *North-Holland mathematics studies*, volume 132, pp. 147--184. Elsevier.

Laporte, G.; Nobert, Y. & Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations research*, 33(5):1050--1073.

Lee, J.-H.; Yu, J.-M. & Lee, D.-H. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, 69(9-12):2081--2089.

Lenstra, J. K. (1977). Sequencing by enumeration methods.

Lenstra, J. K. & Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221--227.

Lenstra, J. K.; Kan, A. R. & Brucker, P. (1977). Complexity of machine scheduling problems. Em *Annals of discrete mathematics*, volume 1, pp. 343--362. Elsevier.

Leung, J. Y. (2004). *Handbook of scheduling: algorithms, models, and performance analysis*. Chapman and Hall/CRC.

Li, F.; Golden, B. & Wasil, E. (2007). A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 34(9):2734--2742.

Liu, L.; Li, W.; Li, K. & Zou, X. (2020). A coordinated production and transportation scheduling problem with minimum sum of order delivery times. *Journal of Heuristics*, 26(1):33--58.

Logendran, R.; McDonell, B. & Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, 34(11):3420--3438.

Lysgaard, J.; Letchford, A. N. & Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423--445.

Mao, J.-y.; Pan, Q.-k.; Miao, Z.-h. & Gao, L. (2021). An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Systems with Applications*, 169:114495.

Martins, L. d. C.; Gonzalez-Neira, E. M.; Hatami, S.; Juan, A. A. & Montoya-Torres, J. R. (2021). Combining production and distribution in supply chains: The hybrid flow-shop vehicle routing problem. *Computers & Industrial Engineering*, 159:107486.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management science*, 6(1):1--12.

Michael, P. (1995). Scheduling. theory, algorithms and systems. *ISBN0-13-706757-7*.

Mohammadi, S.; e Hashem, S. M. A. & Rekik, Y. (2020). An integrated production scheduling and delivery route planning with multi-purpose machines: A case study from a furniture manufacturing company. *International Journal of Production Economics*, 219:347–359.

Monden, Y. (2011). *Toyota production system: an integrated approach to just-in-time*. CRc Press, Boca Raton, FL.

Nagano, M.; Tomazella, C.; Tavares-Neto, R. & Abreu, L. (2022). Solution methods for the integrated permutation flowshop and vehicle routing problem. *Journal of Project Management*, 7(3):155--166.

Nahhas, A.; Kharitonov, A. & Turowski, K. (2022). Deep reinforcement learning techniques for solving hybrid flow shop scheduling problems: Proximal policy optimization (ppo) and asynchronous advantage actor-critic (a3c).

Nawaz, M.; Enscore Jr, E. E. & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91--95.

Nazari, M.; Oroojlooy, A.; Snyder, L. & Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31.

Ngueveu, S. U.; Prins, C. & Calvo, R. W. (2010). An effective memetic algorithm for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 37(11):1877--1885.

Pan, Q.-K.; Gao, L.; Wang, L.; Liang, J. & Li, X.-Y. (2019). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications*, 124:309–324.

Peng, B.; Wang, J. & Zhang, Z. (2020). A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. pp. 636--650.

Penna, P. H. V.; Subramanian, A. & Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19(2):201--232.

Pinedo, M. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing.

Rabadi, G.; Moraga, R. J. & Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17(1):85--97.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M. & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Reina, C. D. (2012). *Roteirização de veículos com janelas de tempo utilizando algoritmo genético.* PhD thesis, Universidade de São Paulo.

Ribeiro, G. M. & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & operations research*, 39(3):728--735.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A. & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254--265.

Sugimori, Y.; Kusunoki, K.; Cho, F. & UCHIKAWA, S. (1977). Toyota production system and kanban system materialization of just-in-time and respect-for-human system. *The international journal of production research*, 15(6):553--564.

Ta, Q. C.; Billaut, J.-C. & Bouquard, J.-L. (2015). Heuristic algorithms to minimize the total tardiness in a flow shop production and outbound distribution scheduling problem. Em *2015 International conference on industrial engineering and systems management (IESM)*, pp. 128--134. IEEE.

Tamannaei, M. & Rasti-Barzoki, M. (2019). Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem. *Computers & Industrial Engineering*, 127:643--656.

Toth, P. & Vigo, D. (2002). *The vehicle routing problem.* SIAM.

Ullrich, C. A. (2013a). Integrated machine scheduling and vehicle routing with time windows. *European Journal of Operational Research*, 227(1):152–165.

Ullrich, C. A. (2013b). Integrated machine scheduling and vehicle routing with time windows. *European Journal of Operational Research*, 227(1):152--165.

Vallada, E. & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612--622.

Wang, S.; Wu, R.; Chu, F. & Yu, J. (2020). Variable neighborhood search-based methods for integrated hybrid flow shop scheduling with distribution. *Soft Computing*, 24(12):8917--8936.

Yağmur, E. & Kesen, S. E. (2024). Integrated production scheduling and vehicle routing problem with energy efficient strategies: Mathematical formulation and metaheuristic algorithms. *Expert Systems with Applications*, 237:121586.

Zar, J. H. (1999). Biostatistical analysis. 4th. *New Jersey, USA*, p. 929.

Zhang, X.; Li, X.-T. & Yin, M.-H. (2020). An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem. *International Journal of Bio-Inspired Computation*, 15(2):113--124.

Zhu, J.; Wang, H. & Zhang, T. (2020). A deep reinforcement learning approach to the flexible flowshop scheduling problem with makespan minimization. Em *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*, pp. 1220--1225. IEEE.

Zou, X.; Liu, L.; Li, K. & Li, W. (2018). A coordinated algorithm for integrated production scheduling and vehicle routing problem. *International Journal of Production Research*, 56(15):5005--5024.

Apêndice A

Trabalhos enviados

A seguir estão relacionados os trabalhos publicados ou em análise para publicação, trabalhos estes resultantes desta tese:

- Artigo em Conferência
  Título: Heuristics Assisted by Machine Learning for the Integrated Production Planning and Distribution Problem.
  Autores: ARAUJO, M. F.; NOGUEIRA, T. H.; ARROYO, J. E.
  Livro: Intelligent Systems Design and Applications - Volume 4
  Evento: 22nd International Conference on Intelligent Systems Design and Applications (ISDA 2022) Held December 12-14, 2022 - Volume 4
  Local: Online.
  Ano: 2022.
  DOI: `https://doi.org/10.1007/978-3-031-35510-3_13`

- Artigo em Conferência
  Título: A Reinforcement Learning Method for Integrated Production Scheduling and Distribution.
  Autores: ARAUJO, M. F.; NOGUEIRA, T. H.; ARROYO, J. E.; ALVES, J. C.
  Livro: Hybrid Intelligent Systems
  Evento: 23nd International Conference on Hybrid Intelligent Systems (HIS

2023) Held December 12-14, 2023
Local: Online.
Ano: 2023.
DOI: Não disponível até a conclusão desta tese.

- Artigo enviado para a revista Operations Research Forum (em analise)
  Título: AI-Enhanced Integration of Production Planning and Vehicle Routing in Logistics Operations: A Hybrid Metaheuristic Framework
  Autores: ARAUJO, M. F.; NOGUEIRA, T. H.; ARROYO, J. E.; ALVES, J. C.
  Revista: Operations Research Forum
  Ano: 2024.

A.0.1 Artigo: Heuristics Assisted by Machine Learning for the Integrated Production Planning and Distribution Problem.

Heuristics assisted by machine learning for the integrated production planning and distribution problem

Matheus de Freitas Araujo, José Elias C. Arroyo and Thiago Henrique Nogueira

Abstract This work addresses a problem that integrates the unrelated parallel machine scheduling and capacitated vehicle routing problems. In this integrated problem, a set of jobs must be processed on machines and then distributed using a fleet of vehicles to customers. The integrated problem's objective is to determine the machines' production scheduling and the vehicle routes that minimize the total weighted tardiness of the jobs. As the problem is NP-Hard, we propose four neighborhood search heuristics and a framework that uses machine learning to solve it. The framework aims to define the best neighborhood search heuristics to solve a given instance based on the problem characteristics. The proposed methods are evaluated and compared by computational experiments on a set of proposed instances. Results show that using a machine learning framework to solve the problem instances yields better performance than neighborhood search heuristics.

1 Introduction

Integrating cyber-physical systems (CPS), the Internet of Things (IoT), and the Internet of Services (IoS) into manufacturing industries will enable the creation of a "smart factory". The idea of a decentralized production system is the basis for the smart factory. This system enables people, machines, and resources to communicate with one another. Smart factories are a fundamental characteristic of Industry 4.0 [4]. Within this context, factories will be able to be fully automated. Autonomous vehicles can carry out ordering systems, production of products, and distribution

Matheus de Freitas Araujo and José Elias C. Arroyo
Departamento de Informática - Universidade Federal de Viçosa, Av. Peter Henry Rolfs, s/n, 36570-900, Viçosa, Minas Gerais, Brazil, e-mail: matheus.f.freitas@ufv.br, jarroyo@ufv.br

Thiago Henrique Nogueira
Universidade Federal de Viçosa, Engenharia de Produção, Rodovia MG-230 - Km 7, 38810-000, Rio Paranaíba, Minas Gerais, Brazil e-mail: thiagoh.nogueira@ufv.br

systems. Automation will make smart factories extraordinarily efficient but also highly flexible and individualized.

New technologies and the advancement of Industry 4.0 have significantly changed production and distribution processes. However, companies should also consider other factors when deciding on customer experience. The approaches mentioned in this paper are geared toward improving customer service levels rather than just minimizing costs. These approaches aim to meet deadlines with minimal delay in an integrated production planning and distribution problem. The problem consists of producing a set of products in unrelated parallel machines and then distributing them to their customers using a fleet of capable vehicles. The objective is to minimize the total weighted tardiness.

Production scheduling and distribution problems are two widely studied combinatorial optimization problems. These problems are usually addressed independently. However, it is possible to find several works that address these integrated problems. Chen [1] [2] presents an extensive review of works addressing these integrated problems. Tamannaei et al. [18] and Felix & Arroyo [3] study an integrated problem that uses a single machine and a heterogeneous fleet of vehicles to deliver items. The objective is to minimize the total weighted tardiness of jobs and transport-related costs. Liu et al. [8], Ngueveu et al. [13] and Ribeiro and Laporte [15] studied a similar problem with the same production system, however they used a homogeneous fleet of vehicles and the objective is to minimize the sum of the jobs' delivery times. Zou [22] studied the integrated problem that uses a single machine and a fleet of vehicles with limited capacity. In this case, the objective is to minimize the maximum delivery time. Ta et al. [17] studied an integrated problem that uses a flow-shop scheduling and a single vehicle with an infinite capacity to carry out deliveries to minimize the total delay.

Naganoa et al. [11], studied an integrated problem of production and distribution in flow-shop scheduling and vehicle routing capacitated problem. The objective is to create a sequence of orders that minimize the integrated problem's makespan, that is, the delivery date from the last job to the last customer. The authors propose a mathematical model of mixed integer programming and an Iterated Greedy algorithm to solve the problem. Martins et al. [10] studied a similar integrated problem, where the production system is modeled as a hybrid flow shop scheduling problem. A vehicle that can carry multiple loads delivers items. The authors propose a mixed integer linear programming model and a metaheuristic biased-randomization variable neighborhood descent (BR-VND) as a solution. Hou et al. [5], developed an enhanced brainstorm optimization (EBSO) algorithm to solve the problem integrated with distributed flow shop scheduling problem and multi-depot vehicle routing optimization problem. The objective is to minimize total weighted earliness and tardiness. The algorithm's results are compared to the mathematical model, and and other works available in the literature [19], [9], [21], [7] [14].

This work proposes a constructive heuristic and four neighborhood search heuristics to solve the production scheduling and distribution problem. Furthermore, we propose a framework that uses machine learning to predict the best heuristic to solve a given instance based on its characteristics. It is essential to look at the character-

istics of the individual instances to determine the most appropriate algorithm as the integrated problem is multi-component, i.e., some instances will have characteristics that will make one of the sub-problems harder to solve than the others.

The paper is organized as follows: Section 2 presents the problem. Section 3 explains the proposed heuristics and the framework. Section 4 presents the computational experiments, and Section 5 presents this work's conclusions.

2 Problem Definition

There is a set of jobs $I = \{1, 2, ..., n\}$ that must be processed, without preemption, on a set $M = \{1, 2, ..., m\}$ of m unrelated parallel machines. Each job j has a processing time p_{ij} on each machine i, a due date (d_j), a tardiness penalty (w_j), and a size (space occupied in a vehicle) (h_j). In addition, there is a machine-dependent setup time s_{ijk} for processing job k just after job j. In general, $s_{ijk} \neq s_{ikj}$.

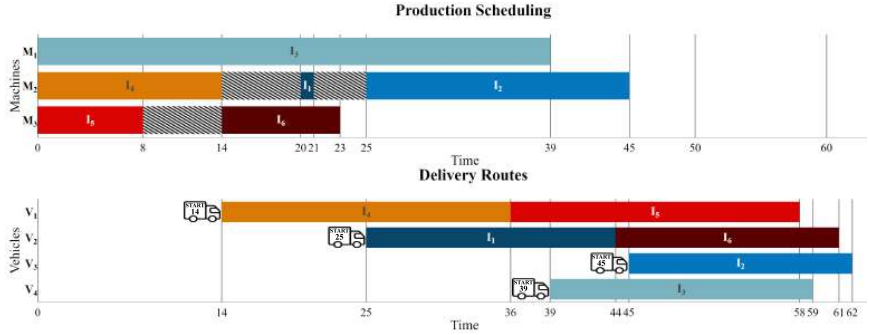
After processing the jobs on the machines, they must be divided into batches and then delivered to their respective clients by using a set $L = \{1, 2, ..., l\}$ of l vehicles. For the routing problem, a graph $G(V, A)$ is defined, where $V = \{0, 1, ..., n\}$ is the set of nodes and $A = \{(j, k) : 0 \geq j, k \geq n, j \neq k\}$ is the set of arcs. The node 0 represents the deposit, and each job is associated with a customer. The set of customers is $I = \{1, ..., n\}$. A travel time t_{jk} is associated with each arc (j, k). Each vehicle $v \in L$ has a capacity (q_v) that must be respected. It is assumed that each vehicle can only be used once, and each customer can only be visited once. The objective of the problem is to determine the processing order of the jobs on the machines and the delivery routes of the used vehicles to minimize the total weighted tardiness (TWT) of the jobs.

The TWT is defined as $\sum_{j \in I} w_j T_j$, where $T_j = max\{0, D_j - d_j\}$ is the tardiness of the job j. D_j is the delivery time of job j. To calculate D_j, first, the processing completion time C_j of each job j and the departure time S_v (start time) of each vehicle v are determined. S_v is defined as $S_v \geq max\{C_j\}$, for all jobs j belonging to the batch of jobs carried by the vehicle v.

To illustrate the problem, in Figure 1, a solution is presented for an instance with three machines ($m = 3$), six jobs ($n = 6$), and four vehicles ($v = 4$).

Figure 1 shows the scheduling of jobs on the machines and the distribution routes. The scheduling of jobs on machines M_1, M_2 and M_3 are [I_5, I_6], [I_4, I_1, I_2] and [I_3], respectively. The delivery routes for vehicles V_1, V_2, V_3 e V_4 are respectively $\{I_0 \rightarrow I_4 \rightarrow I_5 \rightarrow I_0\}$, $\{I_0 \rightarrow I_1 \rightarrow I_6 \rightarrow I_0\}$, $\{I_0 \rightarrow I_2 \rightarrow I_0\}$ and $\{I_0 \rightarrow I_3 \rightarrow I_0\}$, where I_0 represents the deposit. The start times for vehicle trips V_1, V_2, V_3, and V_4 are 14, 25, 45, and 39, respectively. Figure 1 shows the completion times and delivery times. For example, the completion and delivery time of job I_6 are 23 and 61, respectively.

Fig. 1 The scheduling of jobs and delivery routes.



3 Proposed Algorithms

We present two solution-decoding algorithms and a constructive algorithm. Then, we propose four neighborhood search heuristics. Furthermore, we present a framework that uses machine learning to select the best neighborhood search heuristic to solve a problem instance. The algorithms are shown in detail in the following section.

3.1 Decoding Algorithms

Even though our problem has multiple components, we can divide the solution into two parts ($sol = \{s, r\}$), with the first part representing machine scheduling (s) and the second part representing vehicle routing (r). The scheduling s is built from the list of jobs $L1$, where $L1$ represents the order in which the jobs are inserted into the machines by the NEH decoding algorithm. Vehicle routing r is obtained from the list of jobs $L2$, and $L2$ represents the order of job insertion into routing by the PIFH decoding algorithm.

The NEH decoding algorithm is based on the NEH heuristic proposed by [12]. This heuristic is frequently employed in scheduling problems. NEH inserts jobs from the $L1$ list sequentially on the machines. These jobs are assigned to the scheduling position, which minimizes machine completion time. The scheduling s is finally defined.

PIFH is the second decoding algorithm. This algorithm is a modification of Solomon's PIFH heuristic ([16]). The PIFH heuristic is an efficient route construction algorithm. This heuristic employs a greedy criterion to insert jobs into the routes sequentially. The PIFH decoding algorithm inserts jobs into the routes sequentially using the L2 list. Each algorithm iteration inserts a job into the routing to minimize the weighted tardiness (wt). The vehicle routes r are finally defined.

3.2 Initial Solution

Using the decoders NEH and PIFH, an initial solution $sol = \{s, r\}$ is obtained. The job sequencing on the machines is built from a list of jobs $L1$. A list of jobs in $L2$ is used to generate delivery routes. The list $L1$ is built by sorting the jobs in descending order of the sum of the median processing and setup times. The list $L2$ is constructed by sorting the jobs in increasing order of the difference between the due date and the job completion time ($dj - Cj$). Neighborhood search heuristics also use this rule to generate the list $L2$.

3.3 Neighborhood Search Heuristics

To improve the quality of the solutions, we developed four neighborhood search heuristics. Neighborhood search methods use two types of moves: exchange and insertion. The movements are made on the job lists $L1$ and $L2$, which are used by the decoders. The exchange movement in the list consists of exchanging two position elements and creating a new list. The insertion movement involves removing an element from the list and then re-inserting it in a different position in the list. Consider the set $R1(L)$ to be a collection of all neighbors of the list L performing exchange moves and the set $R2(L)$ to be a collection of all neighbors of the list L performing insertion moves.

The first heuristics perform the exchange and insertion movements on the list of jobs $L2$. The Neighborhood Search Heuristics 1 (NSH 1) takes the lists $L1$ and $L2$ as input and performs the following steps:

Step 1: From $L1$ and $L2$, the decoders determine the solution $sol = \{s, r\}$.
Step 2: For each list l in $R_k(L2), \forall k \in \{1, 2\}$. The PIFH decoder sets the route r', building a new solution $sol' = \{s, r'\}$.
Step 3: Check if the new solution is better than the current solution ($F(sol') < F(sol)$).

Step 3.1: If the new solution is better, update solution sol and list $L2$ with l.

Step 4: Return the best solution (sol).

The second Neighborhood Search Heuristics (NSH 2) works similarly to the previous algorithm; however, the exchange and insertion movements are performed on the list $L1$ instead of $L2$. The algorithm begins with the lists $L1$ and $L2$ and follows the steps below:

Step 1: From $L1$ and $L2$, the decoders determine the solution $sol = \{s, r\}$.
Step 2: For each list l in $R_k(L1) \forall k \in \{1, 2\}$. The NEH decoder sets the scheduling s'.
Step 3: The list $L2$ is created from the scheduling s'.
Step 4: The PIFH decoder sets routing r', building a new solution $sol' = \{s', r'\}$.

Step 5: Check if the new solution is better than the current solution ($F(sol') < F(sol)$).

Step 5.1: If the new solution is better, update solution sol and list $L1$ with l.

Step 6: Return the best solution (sol).

The Neighborhood Search Heuristics 3 (NSH 3) algorithm also performs exchange and insertion movements in the list $L1$. The only difference between NSH 3 and NSH 2 is the step 4. Step 4 of the NSH 2 should be replaced by:

Step 4: The Neighborhood Search Heuristics 1 is executed, passing the lists l and $L2$ as parameters. The NSH 1 returns a new solution sol'.

The Neighborhood Search Heuristic 4 (NSH 4) works similarly to the previous NSH 3. The only difference between them is that, in addition to executing step 5.1, whenever a better solution is found, the following steps are also performed:

Step 5.2: Update lists $R_k(L1), \forall k \in \{1, 2\}$.
Step 5.3: Reset the search and return to Step 2 from the beginning.

3.4 Framework

To solve combinatorial optimization problems, the algorithm configuration that produces the best average result for a given set of instances is usually chosen. In general, this algorithm yields good results in some cases and bad results in others. Based on this, we propose a framework that uses machine learning (ML) to determine the best heuristic for solving a specific instance of the problem.

For this purpose, the instances of the problem were initially classified based on their similarities. We define 19 classes of instances. Following categorization, a random instance from each class was chosen to constitute the training set. The remaining instances were used as a test set. The proposed heuristics ran for a maximum of 600 seconds for each instance of the training set, and we chose the heuristic that determined the best solution.

After running all the heuristics for the train set, a model based on ML is built to predict the best heuristic to solve an instance. The ML model has the following instance characteristics as input: number of jobs (n), number of machines (m), average processing time ($\overline{p}$), average setup time ($\overline{s}$), average travel time ($\overline{t}$), average demand ($\overline{h}$) and average vehicle capacity ($\overline{q}$). The ML model output is the best heuristic to solve each specific instance based on its characteristics.

The intention of training the ML model is to build a function $h(x) : x \rightarrow Y$, so that $h(x)$ is predicted to the corresponding value of Y. After the training process, we have a predictor $h(x)$ which, given the characteristics of an instance, returns the best heuristic algorithm structure.

The predictor is a Multilayer Perceptron (MLP) Neural Network with sequential linear layers. Each layer computes its output by using a linear combination of the

inputs. The MLP was trained using the Adam algorithm described in [6]. This algorithm is a first-order gradient-based optimization algorithm for stochastic objective functions based on adaptive estimation of lower-order moments. The Mean Squared Error (MSE) of the differences between the target output and the network output is used to evaluate network performance. The neural network training took about 3 hours. However, once the model is trained, the time spent obtaining the recommended algorithm for any instance is close to 0.1 seconds. As shown in the following section, using this ML framework, it was possible to determine the best solutions for the integrated problem.

4 Computational Experiments

In this section, we present the results of experiments conducted to evaluate the performance of the proposed methods: neighborhood search heuristics and the ML framework. All methods were written in Python 3 and ran on an Intel Core i5-3570 (3.4 GHz) CPU with 16 GB of RAM. The set of instances and all the algorithms presented in this paper are available at `https://github.com/matheusinhofreitas/Machine_Learning_Framework`.

We measure solution quality by the relative percentage deviation (GAP) from the best-known solution obtained among all the methods. The GAP is defined by $GAP = 100 \times \frac{(F_{method} - F_{best})}{F_{method}}$, where F_{best} is the minimum TWT value obtained among all the compared methods and F_{method} is the TWT obtained with a given method.

To carry out the computational experiments, a set of 720 random instances is generated based on some works from the literature. An instance of the problem is characterized by the following parameters: the number of machines $m \in \{2, 4, 8\}$, the number of jobs $n \in \{10, 15, 30, 50\}$, the setup times (generated in the ranges $[1, 9]$ or $[1, 99]$), the average travel time between customers and the depot (20 or 100) and the total vehicle capacity (which is 20%, 50% or 90% higher than total customer demand). Five instances were generated for each parameter combination, totaling 720 instances.

4.1 Computational Results

Initially, the ML Framework (MLF) is compared to the neighborhood search heuristics (NSH 1, NSH 2, NSH 3, and NSH 4). The average GAPs are classified based on the number of jobs and machines. Table 1 values represent the average GAPs and runtime across 60 instances. The smallest GAPs are highlighted in bold.

Table 1 shows that NSH 4 outperforms the other methods for small instances (n = 10, 15 jobs). NSH 4 performs worst in the other instance groups (n = 30, 50). The ML framework obtains the best average GAP for two groups out of a total of 12

Table 1 Average GAPs (%) and runtime (seconds) for the compared methods.

Instances	ML Framework		NSH 1		NSH 2		NSH 3		NSH 4	
	GAP	Time	GAP	Time	GAP	Time	GAP	Time	GAP	Time
n10_m2	13.7	7.75	62.6	0.07	38.1	0.16	09.8	8.94	**04.6**	17.94
n10_m4	05.1	9.67	55.3	0.07	40.4	0.21	05.1	9.12	**03.2**	17.98
n10_m8	03.3	9.66	34.2	0.07	33.0	0.25	03.3	9.04	**02.2**	14.95
n15_m2	18.9	117.23	78.2	0.37	53.3	0.98	17.3	114.22	**09.1**	177.22
n15_m4	11.6	124.18	57.2	0.38	50.6	1.06	12.1	116.12	**03.3**	179.36
n15_m8	4.8	119.65	42.8	0.38	40.4	3.63	4.8	115.13	**01.3**	176.46
n30_m2	19.4	106.86	59.3	6.67	**07.3**	22.47	37.2	180.33	42.4	180.12
n30_m4	**12.3**	167.39	45.9	6.90	23.1	24.51	13.4	180.24	25.2	180.37
n30_m8	06.1	180.35	28.9	7.09	25.3	30.85	**03.8**	180.57	14.6	180.32
n50_m2	07.6	180.53	64.3	57.96	**02.9**	180.25	56.0	180.42	57.7	180.68
n50_m4	**07.0**	180.54	33.0	61.32	07.9	180.06	24.7	180.20	31.4	180.37
n50_m8	09.2	180.56	12.2	151.03	15.1	180.08	**08.6**	180.12	09.5	180.21
Average	**09.9**	115.4	47.8	24.4	28.1	52.0	16.4	121.0	17.0	138.7

groups of instances. The heuristics NSH 1, NSH 2, NSH 3, and NSH 4 produced the best GAP for 0, 2, 2, and 6 groups of instances, respectively. The average GAP for the methods ML Framework, NSH 1, NSH 2, NSH 3, and NSH 4 is 9.9%, 47.8%, 28.1%, 16.4%, and 17%, respectively. The ML Framework is the best method when considering the overall average.

The ML Framework obtained the best GAP for 396 instances (55%) out of 720 instances. In 346 cases (48.1%), the heuristic that found the best solutions was NSH 4. For 345 instances (47.9%), the NSH 3 found the best GAP. The NSH 2 and NSH 1 found the best GAP for 201 (27.9%) and 22 (3.1%), respectively.

In table 1, it is still possible to see that the MLF is similar to the execution time of the NSH3 and NSH4 Algorithms. The only disadvantage of MLF to other algorithms is the need for previous machine learning model training. However, MLF training only needs to be performed once. The MFL can capture the profile of several instances during training. Training is only necessary again if the instances to be resolved by MLF are utterly different from the classes defined in model training.

An Analysis of Variance (ANOVA) [20] was performed to determine whether the observed differences are statistically significant. Using the P-value of the ANOVA, which was 0.0 for the compared methods, and a critical value of 0.05 (5%), it is possible to conclude that there are statistically significant differences between the experiments.

Figure 2 depicts the percentage of times the framework selected each neighborhood search heuristic to solve a given instance (720 instances). The framework only used NSH 1 once (0%) and NSH 2 148 times (21%). The NSH 3 was the most chosen method, 530 times (74%). Finally, the NSH 4 was selected in 41 cases (6%).

Figure 3 shows how often (percentage) the framework chose each heuristic to solve an instance. The ML Framework chose the best heuristic in 57% of the cases and the second best in 34% of the cases. In 91% of cases, the ML Framework decided appropriate heuristics.

Fig. 2 Framework choices

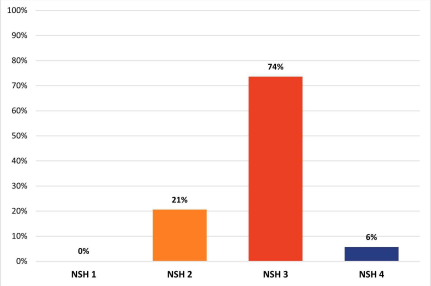
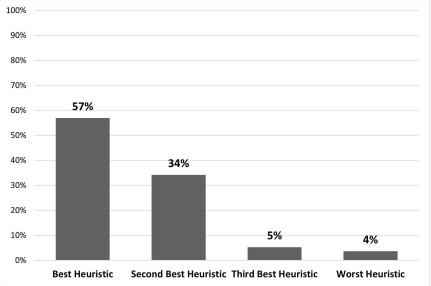


Fig. 3 Framework assertiveness



The Figures 4, 5, 6 and 7 show the average GAP values of the best-known solution for the main characteristics of the instances. The values were divided into intervals, and the average GAP for each resolution method is known. Furthermore, the overall average of the instance results for all methods is displayed. For a given instance, the AVG computes the method dispersion. For cases of low complexity, all methods are expected to find solutions close to the optimal one, i.e., the average value (AVG) will be close to zero, and method dispersion will be low. Only a few methods can perform well against the GAP as complexity increases, resulting in higher dispersion and AVG.

When analyzing the Figures 4, 6 and 7, it is observed that AVG increases with the growth of $\overline{p}$, $\overline{h}$ e $\overline{q}$, while the increase in $\overline{s}$ (Figure 5) interfered slightly with AVG, the increase in complexity is because the more significant the average capacity, the fewer vehicles are available to the instance. The $\overline{s}$ was generated independently of the instance. However, it is still possible to notice a dispersion of the methods with its growth.

The Figures 4 and 5 present the results of the algorithms when comparing the average processing time ($\overline{p}$) and average setup time ($\overline{s}$) of the instances, respectively. In these cases, the MLF proved competitive compared to the other heuristics, achieving good results for different ranges of values for various characteristics. Furthermore, it is possible to observe that, in some intervals, NSH 4 proved to be more efficient than the other heuristics. This happens because the values found by NSH 4, in these cases, determine the F_{best} used to define the GAP.

The Figures 6 and 7 present the results of the algorithms when comparing the average demand ($\overline{h}$) and average vehicle capacity ($\overline{q}$) of the instances. In these cases, the MLF was able to find good results for the different ranges of values of the different characteristics. From the previous graphs, it is possible to observe that the MLF can adapt to changes in the characteristics of the instances. The MLF could recognize and select the best heuristic for each instance.

Fig. 4 Average Processing Time ($\overline{p}$)

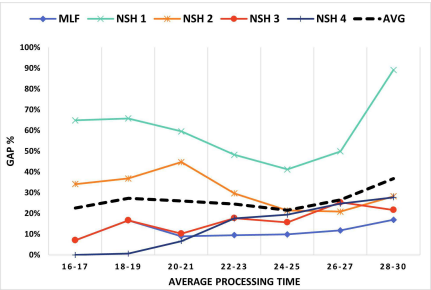


Fig. 5 Average Setup Time ($\overline{s}$)

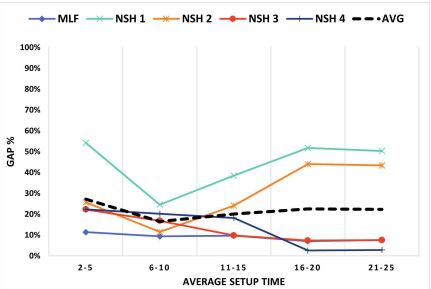


Fig. 6 Average Demand ($\overline{h}$)

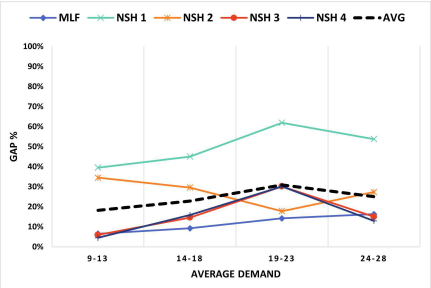
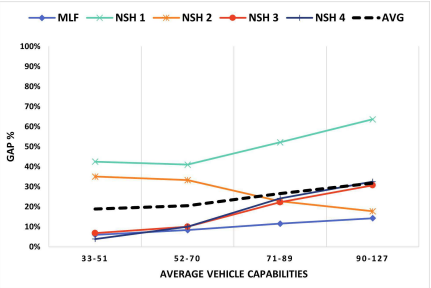


Fig. 7 Average Vehicle Capacity ($\overline{q}$)



5 Conclusions

This paper proposed a MILP model and four neighborhood search heuristics for the integrated production scheduling and distribution problem. We presented a machine learning framework to choose the best heuristic to solve each specific instance of the problem. Computational experiments revealed that the results obtained through the framework were more effective than the resolution of the MILP model and heuristic. Furthermore, changes in the characteristics of the instances had a minimal effect on the framework's results, making this strategy suitable for determining the best algorithm to use to solve each problem instance.

In future studies, we suggest implementing Local Search Algorithms and Iterated Local Search (ILS) to solve this problem. Furthermore, we recommended adapting the Machine Learning Framework to estimate the best local search algorithms to be used by the ILS to solve a given instance. In addition, the Machine Learning Framework should be used in other combinatorial optimization problems. The ML Framework can be used, in these cases, to estimate the best meta-heuristic to solve a given problem instance.

Acknowledgments: This work was supported by CAPES and CNPq.

References

1. Chen, Z.L.: Integrated production and distribution operations. Handbook of quantitative supply chain analysis
2. Chen, Z.L.: Integrated production and outbound distribution scheduling: review and extensions. Operations research **58**(1), 130–148 (2010)
3. Felix, G.P., Arroyo, J.E.C.: Heurísticas para o sequenciamento da produção e roteamento de veículos com frota heterogênea. LII Simpósio Brasileiro de Pesquisa Operacional
4. Hofmann, E., Rüsch, M.: Industry 4.0 and the current status as well as future prospects on logistics. Computers in industry **89**
5. Hou, Y., Fu, Y., Gao, K., Zhang, H., Sadollah, A.: Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows. Expert Systems with Applications **187**
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980**
7. Kurdi, M.: A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem. Applied Soft Computing **94**
8. Liu, L., Li, W., Li, K., Zou, X.: A coordinated production and transportation scheduling problem with minimum sum of order delivery times. Journal of Heuristics **26**(1), 33–58 (2020)
9. Mao, J.y., Pan, Q.k., Miao, Z.h., Gao, L.: An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. Expert Systems with Applications **169**
10. Martins, L.d.C., Gonzalez-Neira, E.M., Hatami, S., Juan, A.A., Montoya-Torres, J.R.: Combining production and distribution in supply chains: The hybrid flow-shop vehicle routing problem. Computers & Industrial Engineering **159**
11. Nagano, M., Tomazella, C., Tavares-Neto, R., Abreu, L.: Solution methods for the integrated permutation flowshop and vehicle routing problem. Journal of Project Management **7**(3), 155–166 (2022)
12. Nawaz, M., Enscore Jr, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega **11**(1), 91–95 (1983)
13. Ngueveu, S.U., Prins, C., Calvo, R.W.: An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Computers & Operations Research **37**(11), 1877–1885 (2010)
14. Pan, Q.K., Gao, L., Wang, L., Liang, J., Li, X.Y.: Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. Expert Systems with Applications **124**
15. Ribeiro, G.M., Laporte, G.: An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & operations research **39**(3), 728–735 (2012)
16. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research **35**(2), 254–265 (1987)
17. Ta, Q.C., Billaut, J.C., Bouquard, J.L.: Heuristic algorithms to minimize the total tardiness in a flow shop production and outbound distribution scheduling problem. In: 2015 International conference on industrial engineering and systems management (IESM)
18. Tamannaei, M., Rasti-Barzoki, M.: Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem. Computers & Industrial Engineering **127**
19. Wang, S., Wu, R., Chu, F., Yu, J.: Variable neighborhood search-based methods for integrated hybrid flow shop scheduling with distribution. Soft Computing **24**(12), 8917–8936 (2020)
20. Zar, J.H.: Biostatistical analysis. 4th. New Jersey, USA p. 929 (1999)
21. Zhang, X., Li, X.T., Yin, M.H.: An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem. International Journal of Bio-Inspired Computation **15**(2), 113–124 (2020)
22. Zou, X., Liu, L., Li, K., Li, W.: A coordinated algorithm for integrated production scheduling and vehicle routing problem. International Journal of Production Research **56**(15), 5005–5024 (2018)

A.0.2 Artigo: A Reinforcement Learning Method for Integrated Production Scheduling and Distribution.

A Reinforcement Learning Method for Integrated Production Scheduling and Distribution

Matheus de Freitas Araujo, Thiago Henrique Nogueira, José Elias Claudio Arroyo and Julio César Alves

Abstract This study addresses an integrate Parallel Machine Scheduling and Capacitated Vehicle Routing problem. In this problem, jobs must be processed efficiently on parallel machines before being distributed to customers through a fleet of vehicles. The main objective of this research is to optimize the machine production scheduling and the vehicle route selection minimizing the job's total weighted tardiness. Since the problem is NP-Hard, we introduce a novel approach using Hybrid Metaheuristics and Reinforcement Learning Algorithms, the PPO-VND, that combines strengths using a Variable Neighborhood Descent Algorithm (VND) and leverages using Proximal Policy Optimization (PPO). The PPO-VND method employs PPO to dynamically select the most effective local search strategy for each iteration within the VND algorithm, evaluated through computational experiments on several instances. The results demonstrate that the VND approach assisted by a Proximal Policy Optimization using a Reinforcement Learning Algorithm outperforms traditional methods such as the Random Variable Neighborhood Descent Algorithm (RVND) and the Mixed-Integer Linear Programming (MILP).

Key words: Machine Learning, Reinforcement Learning, Parallel Machine Scheduling, Vehicle Routing Problem, Hybrid Metaheuristics

Matheus de Freitas Araujo and José Elias Claudio Arroyo
Departamento de Informática - Universidade Federal de Viçosa, Av. Peter Henry Rolfs, s/n, 36570-900, Viçosa, MG, Brazil, e-mail: matheus.f.freitas@ufv.br, jarroyo@ufv.br

Thiago Henrique Nogueira
Universidade Federal de Viçosa, Engenharia de Produção, Rodovia MG-230 - Km 7, 38810-000, Rio Paranaíba, MG, Brazil e-mail: thiagoh.nogueira@ufv.br

Julio César Alves
Universidade Federal de Lavras, Departamento de Computação Aplicada, Trevo Rotatório Professor Edmir Sá Santos, s/n, 37200-900, Lavras, MG, Brazil e-mail: juliocesar.alves@ufla.br

1 Introduction

Production and distribution methods have radically changed due to the Industry 4.0 emergence [9], requiring companies to operate based on parameters other than optimization processes or cost reduction. Therefore, this study aims to improve customer service quality rather than reduce costs. The authors attempt to solve the integrated production, planning, and distribution problems as efficiently and quickly as possible, adopting as a research problem the assemblage of products on unrelated parallel machines with sequence-dependent setup times that, once created, are sent to customers using a fleet of capable vehicles. Also, the research's general goal is to minimize the total weighted delay.

Mathematical models, heuristics, and metaheuristics commonly apply to combinatorial optimization problems, with several pieces of research proposing hybrid approaches. Hybrid algorithms do not purely follow the paradigm of a single traditional heuristic or metaheuristic, using distinct parts of different optimization techniques to find the best solution. For instance, Blum et al. [2] presented a Hybrid Metaheuristics (HM) survey investigating the complementary nature of different hybrid strategies.

Chen [3] [4] provides an extensive literature review on integrated production scheduling and distribution problems. Also, Liu et al. [11] examined a single-machine scheduling and vehicle routing problem to minimize the totality of a job's delivery time using a Variable Neighborhood Search (VNS) heuristic, later comparing the VNS-produced results to those obtained with a Memetic Algorithm (MA) [13] and a Large Neighborhood Search Algorithm (LNS) [15].

Tamannaei et al. [17] proposed a mathematical model, a B&B algorithm, and a genetic algorithm to minimize the total weighted job delay and the transportation costs in a single-machine scheduling and capable vehicle routing problem. Likewise, Felix & Arroyo [6] proposed two hybrid metaheuristics based on ILS and RVND to solve the same problem, comparing their results with Tamannaei et al.'s.

Zhu et al. [19] proposed a Deep Reinforcement Learning (DRL) approach to solve a Flexible Flow Shop Scheduling Problem (FFSP) with makespan minimization to reduce the time required to complete a set of jobs on a series of identical parallel machines. The authors adopted Proximal Policy Optimization (PPO) to evaluate and optimize random and complex instances with different scales, including real-world instances. They established that the PPO always provided satisfactory solutions employing reasonable computational time.

Nahhas et al. [12] applied Deep Reinforcement Learning (DRL) techniques to optimize Hybrid Flow Shop (HFS) scheduling. The DRL techniques, Proximal Policy Optimization (PPO), and Asynchronous Advantage Actor-Critic (A3C) proved effective in learning appropriate policies for solving HFS scheduling and adjusting their strategies to different problems.

Kayhan and Yildiz [10] presented a comprehensive review of Reinforcement Learning (RL) applications for machine scheduling problems, analyzing algorithms, machine environments and characteristics, jobs, objectives, and reference methods. They indicated that the most frequently investigated topics are Job-Shop Scheduling Problems, Unrelated Parallel Machine Scheduling, and Single-Machine Schedul-

ing. Their main contributions are thoroughly examining RL in machine scheduling, categorizing recurring problem types, objectives, and constraints, and divulging literature gaps.

In this study, we propose solving the abovementioned problem using a Mixed Integer Linear Programming Model (MILPM), a Constructive Heuristic, and a PPO-VND Hybrid Metaheuristic that combines VND with a Reinforcement Learning Algorithm and a Proximal Policy Optimization (PPO) algorithm, comparing the PPO-VND results with the MILPM and the Random Variable Neighborhood Search (RVND) metaheuristic, a variation of VND with a random neighborhood option.

This paper's organization is as follows: Sections 2 and 3 state the research problem and describe the mathematical model, respectively. Section 4 presents the Constitutive Heuristics, the Proximal Policy Optimization, and the proposed Hybrid Metaheuristic. In Section 5, the authors demonstrate the computational experiments conducted for this research. Finally, Section 6 concludes the study.

2 Problem Definition

There is a set of jobs $I = \{1, 2, ..., n\}$ that must be processed, without preemption, on a set $M = \{1, 2, ..., m\}$ of m unrelated parallel machines. Each job j has a processing time p_{ij} on each machine i, a due date (d_j), a tardiness penalty (w_j), and its size (space occupied in a vehicle) (h_j). Also, there is a machine-dependent setup time s_{ijk} for processing the job k, just after job j. In general, $s_{ijk} \neq s_{ikj}$.

After processing the jobs on the machines, they are split into batches, each delivered to the respective clients using a set $L = \{1, 2, ..., l\}$ of l vehicles. For the routing problem, graph $G(V, A)$ displays a $V = \{0, 1, ..., n\}$ the set of nodes and an $A = \{(j, k) : 0 \geq j, k \geq n, j \neq k\}$ set of arcs. The node 0 represents the deposit, and each job is associated with a customer, being the set of customers $I = \{1, ..., n\}$. Furthermore, a travel time t_{jk} is associated with each arc (j, k). Each vehicle $v \in L$ has a capacity (q_v) that must be respected. The problem assumes single-use vehicles and one visit per customer. The problem aims to determine the jobs' processing orders and the vehicles' delivery routes to minimize the jobs' total weighted tardiness (TWT).

Figure 1 illustrates the issue and presents a solution for an example with three machines ($m = 3$), six jobs ($n = 6$), and four vehicles ($v = 4$). Figure 1 shows each job's completion times, the delivery times for each job, and the vehicle routes. For instance, the completion and delivery time of job I_4 are 14 and 36, respectively. The vehicle V_1 starts at 14 and is responsible for delivering jobs I_4 and I_5, following the route: $\{I_0 \rightarrow I_4 \rightarrow I_5 \rightarrow I_0\}$, where I_0 represents the deposit.

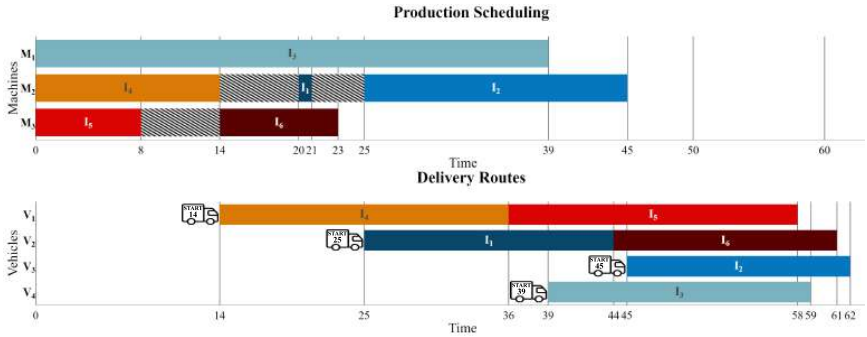


Fig. 1: The scheduling of jobs and delivery routes.

3 Mixed Integer Linear Programming Model

The authors propose a Mixed Integer Linear Programming (MILP) model to solve the problem. This model uses the parameters in the problem description and the following decision variables: X_{ijk} is a binary variable equal to '1' if job j precedes job k on machine i. The Y_{ij} is a binary variable equal to '1' machine i processes the job j. The Z_{vjk} is a binary variable equal to '1' if the vehicle v travels between points j and k. The U_v is a binary variable equal to '1' if the vehicle v is used for deliveries. The C_j is the job j's completion time. The S_v is the vehicle's start time v. The T_j is the job j's delivery delay, and D_j is the job j's delivery time.

The MILP model also requires the following parameters: $N = I \cup \{O\}$ represents a set containing the jobs and a dummy job starting from the origin, and MG is a sufficiently large value defined by $\sum_{i \in M} \sum_{j \in N} \sum_{k \in N} t_{jk} + p_{jk} + s_{ijk}$. The mathematical model is presented below. The objective of the model is:

$$\min \ \text{TWT} = \sum_{j \in I} w_j T_j \tag{1}$$

$$\sum_{v \in L} \sum_{k \in N, k \neq j} Z_{vjk} = 1, \quad \forall j \in I \tag{2}$$

$$\sum_{j \in N} \sum_{k \in N, k \neq j} Z_{vjk} h_j \leq q_v U_v, \quad \forall v \in L \tag{3}$$

$$\sum_{k \in N} Z_{v0k} = U_v, \quad \forall v \in L \tag{4}$$

$$\sum_{j \in N} Z_{vjh} = \sum_{k \in N} Z_{vhk} \forall h \in N, \quad \forall v \in L \tag{5}$$

$$\sum_{k \in I} X_{i0k} \leq 1, \quad \forall i \in M \tag{6}$$

$$\sum_{i \in M} Y_{ij} = 1, \quad \forall j \in I \tag{7}$$

$$Y_{ij} = \sum_{k \in N, k \neq j} X_{ijk}, \quad \forall i \in M, \forall j \in I \tag{8}$$

$$Y_{ik} = \sum_{j \in N, k \neq j} X_{ijk}, \quad \forall i \in M, \forall k \in I \tag{9}$$

$$C_k - C_j + MG(1 - X_{ijk}) \geq s_{ijk} + p_{ik}, \quad \forall j \in N, \forall k \in I, j \neq k, \forall i \in M \tag{10}$$

$$S_v \geq C_k - MG(1 - \sum_{j \in N, j \neq k} Z_{vjk}), \quad \forall v \in L, \forall k \in N \tag{11}$$

$$D_k - S_v \geq t_{0k} - MG(1 - Z_{v0k}), \forall v \in L, \qquad \forall k \in N \tag{12}$$

$$D_k - D_j \geq t_{jk} - MG(1 - \sum_{v \in L} Z_{vjk}), \quad \forall j \in I, \forall k \in N, j \neq k \tag{13}$$

$$T_j \geq D_j - d_j, \quad \forall j \in I \tag{14}$$

The objective function, defined by Equation (1), is designed to minimize the sum of the weighted job delays. The constraints (2) ensure that each customer is served only once and by a single vehicle. The restrictions (3) guarantee the maintenance of the vehicles' capacities, considering $h_0 = 0$. The constraints (4) and (5) assure that, when used, a vehicle must start and end its route at the depot. Constraints (6) ensure that only one job runs after each machine's dummy job I_0. Constraints (7) guarantee that each job runs on only one machine. Constraints (8) and (9) provide that each job has only one successor and one predecessor. Constraints (10) define the job completion time, considering $C_0 = 0$. The Constraints (11) define each travel's starting time. The vehicle arrival times are according to the constraints (12) and (13). Finally, constraints (14) define job delays based on due dates.

4 Proposed Algorithms

This section presents the Proximal Policy Optimization (PPO), Reinforcement Learning, and a Constructive Algorithm. Following, the authors propose the PPO-VND Hybrid Metaheuristic that incorporates the Variable Neighborhood Algorithm (VND) aided by the Reinforcement Learning Algorithm, PPO.

4.1 Initial Solution

To obtain an initial solution ($solution = \{s, r\}$), defining the job list $L1$ is necessary. The job list $L1$ ranks the jobs in descending order according to the sum of average processing and setup times. Subsequently, the NEH algorithm uses the job list $L1$ to define the machines' scheduling. The NEH algorithm sequentially inserts the jobs from the $L1$ list into the solution. At each algorithm iteration, a job $j \in L1$ is inserted into the machine's sequencing (s) to minimize the machine's completion time.

Defining the job list $L2$ is necessary to build the delivery routes since the $L2$ list comes from scheduling jobs using the Earliest Due Date (EDD) rule ($d_j - C_j$). In this case, jobs with more immediate due dates receive their routes first from the PIFH algorithm. The PIFH algorithm sequentially inserts the list of jobs $L2$ to minimize the weighted delay (wt). At the end of the algorithm, the vehicle routes r are finally defined. Araujo et al.[7]'s research provides more information on NEH and PIFH heuristics.

4.2 The Proximal Policy Optimization

Reinforcement Learning (RL) commonly applies to sequential decision-making problems and consists of an agent interacting with a stochastic environment, learning to map states into actions and maximizing cumulative rewards. The RL problem can be modeled as a Markov Decision Process (MDP), a tuple $\{S, A, p, r(s, a)\}$, where S is the set of states, A is the set of actions, and p is the transition function that represents the conditional probabilities of state change (between 0 and 1). Ultimately, $r(s, a)$ is the reward function that assigns a numerical value to the state-action pair. In these processes, the agent aims to find an approach that associates each state with an optimal action, maximizing the sum of rewards over time [1].

The Proximal Policy Optimization (PPO) algorithm, proposed by Schulman et al. [16], derives from the concept of Policy Gradients (PG) and attempts to directly optimize agents' policies through an on-policy approach, using a Deep Neural Network (DNN) to map actions to states. PPO trains the DRL agents to act in an environment, maximizing the cumulative reward and directly finding the best policy rather than learning the function value and to derive the best one.

Actor-critical are GP methods that approximate the policy and value function. The actor learns the policy, while the critic learns the value function to evaluate the stated policy. This approach speeds up the learning process by minimizing variation during training. Proximal Policy Optimization (PPO) balances implementation, parameterization, and sample complexity. Its main goal is to limit step size updates, revamping the policy with a new one similar to the current one. This process is important because large updates cause significant variation in the learning process, leading to poor performance [1]. The authors use the PPO algorithm from the stable-baselines3 library in Python [14]. The PPO pseudocode is presented in Algorithm 1.

Algorithm 1 Proximal Policy Optimization

Initialize Θ and Φ ▷ policy and value-function parameters, respectively
for $i = 0, 1, 2, \ldots$ **do**
  for $actor = 1, 2, \ldots, N_{actors}$ **do**
    Run policy $\pi(\Theta)$ for T steps and collect the trajectories.
    Calculate advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ based on value-function V_Φ.
  end for
  Form a *batch*, of size NT , with collected trajectories and advantages.
  for $epoch = 1, 2, \ldots, K$ **do**
    Shuffle *batch* and split into *minibatches*
    for all *minibatches* **do**
      Update Θ wrt objective function via stochastic gradient.
      Update Φ wrt mean-squared error of value-function V_Φ.
    end for
  end for
end for

4.2.1 State, Action, and Reward Function

The state is the data that fully represents the current environment that contains the agent. The state's portrayal is a vector feature that describes the environment at a given time: a solution to the problem containing the number of machines (m), the number of jobs (n), the list of the jobs' completion times ($C_j, \forall j \in N$), the list of jobs' delivery dates ($D_j, \forall j \in N$), and the list of tardiness penalties ($w_j T_j, \forall j \in N$).

The action is the decision taken by the agent in a given state and represents the agent's next choice. In this case, the choice is the next neighborhood explored by the local search. The model output is a $\sum_{j \in I} w_j T_j$ of solutions created from job lists $L1$ and $L2$ after the action taken by the agent.

The reward function comes from the objective function, calculated using the equation 15 and considering the lower bound (LB) and upper bound (UB). LB is calculated using the equations 16. To calculate UB, the authors assume that the jobs' distribution will be identical among machines and vehicles. The authors also consider that job j will be the last executed on the machine and delivered on the route. Furthermore, all jobs have the longest possible processing, setup, and travel times.

$$\text{Reward Function} = -1 * (\sum_{j \in I} w_j T_j) - LB)/(UB - LB) * 100 \tag{15}$$

$$LB = \sum_{j \in I} w_j * Max((Min(p_{ij} \forall i \in M) + t_{0j} - d_j), 0) \tag{16}$$

4.2.2 The Environment Model

An agent's interaction with the environment demands an environmental model to describe the problem. The environment depicts the Integrated Production-Distribution

Planning Problem. The model inputs consist of two job lists ($L1$ and $L2$) used to resolve said problem ($solution = \{s, r\}$). The model's output is the objective function that defines the criterion for evaluating the solution. The agent aims to find an approach that maximizes cumulative rewards over time, leading to better solutions for the integrated problem. This research problem required creating an environmental model for each group of instances with n equal to 10 and 15, as the environmental model directly links to the number of jobs. The Environment Model appears in Algorithm 2.

Algorithm 2 Environment Model

1: $solution$ = New solution$\{s, r\}$.
2: Sort the job sequence as $L1$ in in descending order of the sum of average processing and setup times.
3: **for** $j \in L1$ **do**
4: Insert j in s to minimize the machine's completion time
5: **end for**
6: Sort the job sequence as $L2$ the earliest due date (EDD) rule ($d_j - C_j$).
7: **for** $j \in L1$ **do**
8: Insert j in r to minimize the job's weighted delay (wt)
9: **end for**
10: Create $solution$ with s and r
11: Return $f(solution)$

4.3 PPO - VND

The Variable Neighborhood Descent (VND) is a deterministic local search algorithm [8] that employs a series of neighborhood structures $V = V_1, V_2, ..., V_k$ until each solution S becomes locally optimal. The PPO-VND hybrid metaheuristic is presented in the algorithm 3. A Proximal Policy Optimization Algorithm (PPO) predicts and defines the explored neighborhood at each algorithm iteration.

The algorithm starts with an initial solution, and at each algorithm iteration, a neighborhood structure u is chosen by the PPO. Then, a local search occurs within the chosen neighborhood u, returning a local optimal solution. If the new solution is better than the "best_solution", the PPO updates it. Otherwise, the algorithm perturbates the solution. The process repeats until the number of non-improving iterations equals n (number of instance jobs) or the solution found equals the Lower Bound (LB).

In this paper, the authors propose eight neighborhoods. The neighborhood methods use two types of actions: insertion and exchange with and without restarting. The movements occur within the job lists $L1$ and $L2$. The insertion implicates removing an element from the list and re-inserting it in a different position. The exchange movement involves exchanging two elements' positions and creating a new list. In addition, the neighborhoods with a restart option will restart the searches whenever a better solution occurs.

The solution's perturbation consists of randomly exchanging and inserting movements in the job lists. The perturbation happens in up to 10% of the jobs' list, i.e., 10%, with the list selection ($L1$, $L2$, or both) occurring randomly. Following, the algorithm 2 uses these job lists ($L1$ and $L2$) to define a new solution, later employed in the algorithm.

Algorithm 3 PPO - VND

Create $L1$ and $L2$,
Create $solution$ using the NEH and PIFH Algorithms
$best\_solution \leftarrow solution$
state $\leftarrow$ initial state condition
while $it < n$ or $f(solution)! = LB$ **do**
    $u \leftarrow$ PPO.predict($state$) ▷ PPO chooses a neighborhood structure u for current state.
    $solution \leftarrow$ Best Solution in $N_u(solution)$
    state $\leftarrow$ update
    if $F(solution) < F(best\_solution)$ **then**
        $best\_solution \leftarrow solution$
        $it \leftarrow 0$
    else
        $it \leftarrow it + 1$
        perturbation the current $solution$
    end if
end while
return $best\_solution$

5 Computational Experiments

In this section, the authors present the experiment to evaluate the performance of the proposed methods. The Mixed-Integer Linear Programming Model (MILP) was implemented in C++ and solved using the CPLEX solver [5]. The other methods were programmed in Python 3 and executed on an Intel Core i7-4790K (4.0 GHz) CPU with 32 GB of RAM. The algorithms presented in this paper are available at `https://github.com/matheusinhofreitas/PPO-VND`. The computational experiments occurred in a single thread with the MILP model processing time fixed at 180 seconds. The authors trained the agent for 500 thousand episodes with 36 randomly selected instances of each size and different complexities. The PPO hyperparameters were tuned using the RL Baselines3 Zoo library. Additionally, the authors developed a Random Variable Neighborhood Search algorithm (RVND) (similar to PPO-VND) that randomly selects the neighborhood structure at each algorithm iteration.

The solution comes from the best-known solution's relative percentage deviation (GAP). The $GAP = 100 \times \frac{(F_{method} - F_{best})}{F_{method}}$, where F_{best} is the minimum TWT value obtained among all the compared methods and F_{method} is the TWT obtained with a given method.

The authors employed the 360 instances proposed by Araujo et al. [7] to conduct the computational experiments. Each problem instance is characterized by two key parameters: the number of jobs denoted as $n \in \{10, 15\}$, and the number of machines represented by $m \in \{2, 4, 8\}$. For example, the instances "N_10_M_2" have 10 jobs and 2 machines.

5.1 Computational Results

Initially, the authors compare the MILP model to the PPO-VND and the RVND. For each instance, they run the PPO-VND and RVND Algorithms five times, classifying the average GAPs based on the number of jobs and machines. Table 1 represents the averages, the best GAPs, and the runtime across 60 instances. The smallest average GAPs are in bold.

Table 1: Average and best GAPs (%) and runtime for the compared methods.

Instance	Average GAP (%)				Best GAP(%)		Time in seconds		
	Initial Solution	**MILPM**	**PPO-VND**	**RVND**	**PPO-VND**	**RVND**	**MILPM**	**PPO-VND**	**RVND**
N_10_M_2	85%	34%	**23%**	46%	12%	25%	173.22	**1.46**	1.53
N_10_M_4	80%	**10%**	34%	38%	26%	22%	173.22	**1.45**	1.54
N_10_M_8	65%	**14%**	18%	17%	12%	5%	173.20	**1.44**	1.54
N_15_M_2	89%	79%	**22%**	53%	9%	31%	173.19	**1.43**	1.54
N_15_M_4	77%	59%	**18%**	31%	8%	10%	172.95	**1.44**	1.55
N_15_M_8	68%	79%	**18%**	19%	7%	5%	172.94	**1.45**	1.55
Average	77%	46%	**22%**	34%	12%	17%	173.12	**1.45**	1.54

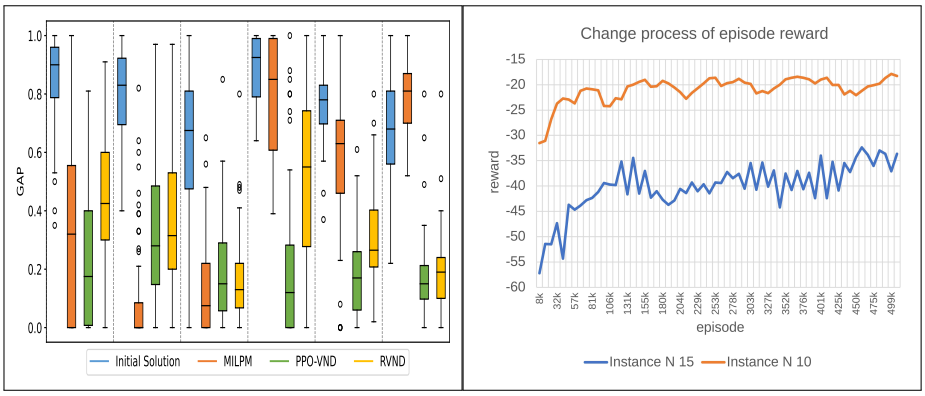
Table 1 shows that the MILPM obtains the best average GAP for two groups out of 6 (N_10_M_4 and N_10_M_8 with the respective values: 10% and 14%), the PPO-VND produced the best GAP for 4 instance groups (N_10_M_2, N_15_M_2, N_15_M_4, N_15_M_8 with the respective values: 23%, 22%, 18% and 18%), and the RVND did not display the best GAP for any group. The average GAP for the MILPM, PPO-VND, and RVND methods is 46%, 22%, and 34%, respectively. Considering the overall average, the PPO-VND is the best method.

Considering all instances, the MILPM obtained the best GAP for 87 (24.1%) out of 360 instances. In 170 cases (47.2%), the PPO-VND found the best solutions. Finally, the RVND found the best GAP for 165 instances (45.8%). Furthermore, the MILPM managed to find the optimal solution for 18 instances.

Table 1 also shows that the execution time of the PPO-VND is similar to that of the RVND algorithm. The only drawback of the PPO-VND is that its algorithm requires prior training using the Deep Reinforcement Learning (DRL) model. However, it is a single-time training process; subsequently, the forecast time is instantaneous.

The authors performed an Analysis of Variance (ANOVA) [18] for a 5% significance level. The ANOVA yielded a p-value of 0.000102, indicating statistically

significant differences between the experiments. Figure 2a shows the boxplot for the GAPs obtained using all methods. The constructive solution is limited when compared to other methods. Additionally, PPO outperforms RVND and rivals the mathematical model, providing equal or superior solutions. The boxplot illustrates each method in different colors, with the initial solution in blue, the MILPM in orange, the PPO-VND in green, and the RVND in yellow. Figure 2b shows the constant increase in the average reward throughout the model training, and the graph indicates a successful learning process for DRL agents to solve the adopted problem. The orange and blue lines represent the reward-changing process for instances with $n = 10$ and $n = 15$ sizes, respectively.



(a) BoxPlot of GAPs (%) for all methods (b) Change process of episode reward for PPO

Fig. 2: General computational results for all methods

6 Conclusions

This paper proposed a Mixed-Integer Linear Programming Model and the Hybrid Heuristics PPO-VND for an integrated production scheduling and distribution problem. The authors applied the Proximal Policy Optimization (PPO) Reinforcement Learning (RL) algorithm to predict the neighborhood structure for each PPO-VND interaction. The computational experiments revealed that the PPO-VND was more effective than the mathematical model and the RVND algorithm concerning execution time and solution quality. In this case, the authors disregarded the training time for the PPO since it only occurred once. It is essential to highlight that even in small instances with small search spaces, such as those used in this experiment, the PPO-VND algorithm obtained better results than the other resolution techniques presented. Compared with the other methods, PPO-VND results improved when the

number of jobs increased from 10 to 15. The authors expect the PPO-VND algorithm to perform even better on larger instances and strive to apply the PPO-VND algorithm to solve such problems in future studies.

Acknowledgments: This work was supported by CAPES and CNPq.

References

1. Alves, J.C., Mateus, G.R.: Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. In: International Conference on Computational Logistics, pp. 584–599. Springer (2020)
2. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied soft computing **11**(6), 4135–4151 (2011)
3. Chen, Z.L.: Integrated production and distribution operations. Handbook of quantitative supply chain analysis pp. 711–745 (2004)
4. Chen, Z.L.: Integrated production and outbound distribution scheduling: review and extensions. Operations research **58**(1), 130–148 (2010)
5. Cplex, I.I.: V12. 1: User's manual for cplex. International Business Machines Corporation **46**(53), 157 (2009)
6. Felix, G.P., Arroyo, J.E.C.: HeurÍsticas para o sequenciamento da produÇÃo e roteamento de veÍculos com frota heterogÊnea. LII Simpósio Brasileiro de Pesquisa Operacional (2020)
7. de Freitas Araujo, M., Arroyo, J.E.C., Nogueira, T.H.: Heuristics assisted by machine learning for the integrated production planning and distribution problem. In: International Conference on Intelligent Systems Design and Applications, pp. 120–131. Springer (2022)
8. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European journal of operational research **130**(3), 449–467 (2001)
9. Hofmann, E., Rüsch, M.: Industry 4.0 and the current status as well as future prospects on logistics. Computers in industry **89**, 23–34 (2017)
10. Kayhan, B.M., Yildiz, G.: Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. Journal of Intelligent Manufacturing pp. 1–25 (2021)
11. Liu, L., Li, W., Li, K., Zou, X.: A coordinated production and transportation scheduling problem with minimum sum of order delivery times. Journal of Heuristics **26**(1), 33–58 (2020)
12. Nahhas, A., Kharitonov, A., Turowski, K.: Deep reinforcement learning techniques for solving hybrid flow shop scheduling problems: Proximal policy optimization (ppo) and asynchronous advantage actor-critic (a3c) (2022)
13. Ngueveu, S.U., Prins, C., Calvo, R.W.: An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Computers & Operations Research **37**(11), 1877–1885 (2010)
14. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021). URL `http://jmlr.org/papers/v22/20-1364.html`
15. Ribeiro, G.M., Laporte, G.: An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & operations research **39**(3), 728–735 (2012)
16. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
17. Tamannaei, M., Rasti-Barzoki, M.: Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem. Computers & Industrial Engineering **127**, 643–656 (2019)
18. Zar, J.H.: Biostatistical analysis. 4th. New Jersey, USA p. 929 (1999)
19. Zhu, J., Wang, H., Zhang, T.: A deep reinforcement learning approach to the flexible flowshop scheduling problem with makespan minimization. In: 2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS), pp. 1220–1225. IEEE (2020)

A.0.3 Artigo (em Revisão): AI-Enhanced Integration of Production Planning and Vehicle Routing in Logistics Operations: A Hybrid Metaheuristic Framework

AI-Enhanced Integration of Production Planning and Vehicle Routing in Logistics Operations: A Hybrid Metaheuristic Framework

Matheus de Freitas Araujo[1*], Thiago Henrique Nogueira[2†], José Elias Claudio Arroyo[1†], Julio César Alves[3†]

[1*]Departamento de Informática, Universidade Federal de Viçosa, Av. Peter Henry Rolfs, s/n, Viçosa, 36570-900, Minas Gerais, Brazil.
[2]Departamento de Engenharia de Produção, Universidade Federal de Viçosa, Rodovia MG-230 - Km 7, s/n, Rio Paranaíba, 38810-000, Minas Gerais, Brazil.
[3]Departamento de Computação Aplicada, Universidade Federal de Lavras, Trevo Rotatório Professor Edmir Sá Santos, s/n, Lavras, 37200-900, Minas Gerais, Brazil.

*Corresponding author(s). E-mail(s): matheus.f.freitas@ufv.br;
Contributing authors: thiagoh.nogueira@ufv.br; jarroyo@ufv.br; juliocesar.alves@ufla.br;
†These authors contributed equally to this work.

Abstract

Integrating production planning on unrelated parallel machines with the vehicle routing problem in modern logistics operations poses significant challenges. Addressing this integration is crucial for optimizing operational efficiency and meeting customer demands. This study proposes a comprehensive approach comprising three methods: a Mixed Integer Linear Programming Model (MILPM), a Constructive Heuristic, and a Neighborhood Search Heuristics (NSH). Additionally, a novel framework that incorporates machine learning predicts the most effective heuristic for each instance. Furthermore, we introduce a pioneering RVND Metaheuristic and Hybrid Metaheuristic (PPO-VND), merging Variable Neighborhood Descent (VND) with Reinforcement Learning Algorithm, Proximal Policy Optimization (PPO). These innovations significantly enhance solution quality and computational efficiency. Moreover, our work underscores the pivotal

role of Artificial Intelligence (AI) in logistics optimization, fostering adaptability and resilience in dynamic operational environments. Comparative tests demonstrate the superiority of our proposed methods over existing approaches, confirming their effectiveness in addressing the critical challenges posed by integrated production planning and vehicle routing in logistics operations.

Keywords: Integrated Production Planning Problem, Parallel Machine Scheduling, Vehicle Routing Problem, Hybrid Metaheuristic, Unsupervised Learning, Reinforcement Learning

1 Introduction

The complexity of contemporary logistics operations, driven by the increasing demand for efficiency and rapid service delivery, has sparked a notable interest in leveraging combinatorial optimization to tackle integrated production and distribution planning challenges. The emergence of Industry 4.0, characterized by the integration of cyber-physical systems (CPS), the Internet of Things (IoT), and the Internet of Services (IoS), has ushered in decentralized production systems, enabling seamless communication among people, machines, and resources. This paradigm shift underscores the critical role of Artificial Intelligence (AI) in optimizing logistics processes. AI enhances resource optimization and cost reduction and elevates service quality within decentralized systems, where any disruptions or delivery delays can lead to substantial losses and supply chain interruptions.

In this context, AI-driven predictive analytics and real-time decision-making capabilities are indispensable for fortifying resilience and ensuring uninterrupted operations amidst the complexities of Industry 4.0. By prioritizing enhancing service quality, organizations can navigate the challenges posed by decentralized production systems, fostering innovation and customer satisfaction. The symbiotic relationship between AI and Industry 4.0 paves the way for dynamic logistics ecosystems where efficiency, reliability, and adaptability converge to meet the evolving demands of the modern marketplace.

Integrated production and distribution planning problems, extensively studied and exemplified by researchers such as Chen [1, 2] and practical applications showcased by Ullrich [3], Yagmur and Kesen [4], Mohammadi et al. [5], Boudia et al. [6], and Tamannaei and Rasti-Barzoki [7], are at the forefront of addressing contemporary logistics complexities. In the context of Industry 4.0, where the Just in Time (JIT) system emerges as a solution to the challenges of operating without stock, seamless connectivity throughout the entire supply chain is imperative. The main idea of the JIT system is to create only what is necessary when it is necessary and with the appropriate resources. This approach helps minimize excess inventory and lowers storage costs. However, its mastery hinges upon the interconnectedness, alignment, and readiness of the entire supply chain to fulfill demands promptly and efficiently, underscoring the pivotal role of Artificial Intelligence (AI) in optimizing logistics operations within the Industry 4.0 framework.

The principles of Just in Time (JIT) production, exemplified by Toyota's pioneering adoption and subsequent success in the automotive industry [8–10], underscore the tangible benefits of streamlined operations within manufacturing and assembly industries. As JIT systems minimize inventories, waste, and idle time while enhancing efficiency and operational effectiveness, they align seamlessly with the challenges of contemporary logistics highlighted in the preceding paragraph. In the era of Industry 4.0, where decentralized production systems demand interconnectedness and agility, JIT principles resonate deeply, emphasizing the need for synchronized supply chains and optimized processes. This symbiotic relationship between JIT methodologies and the logistics landscape underscores the vital role of Artificial Intelligence (AI) in orchestrating seamless operations, driving efficiency, and ensuring the timely delivery of goods and services amidst evolving complexities.

Our study delves into the intricacies of decentralized production systems, where the seamless integration of production planning and distribution is paramount. Unlike conventional methodologies primarily focused on cost reduction and resource optimization, our research shines a spotlight on enhancing customer service levels. By prioritizing the minimization of total weighted delay in product delivery to customers, our study aligns with the evolving demands of contemporary logistics operations. Leveraging insights from JIT principles and AI-driven optimization, we navigate the complexities of Industry 4.0, fostering synchronized supply chains and driving operational efficiency. Through a holistic approach that emphasizes service quality and responsiveness, our research contributes to the ongoing discourse on optimizing logistics operations within the dynamic landscape of Industry 4.0.

We provide an overview featuring articles related to integrating production planning with the vehicle routing problem in Table 1. Drawing on the insights from various authors who have explored integrated production and distribution planning systems, our study delves into the challenges posed by decentralized production systems. Nagano et al.[11] contribute to this discourse by investigating an integrated production and distribution problem within a flow shop system and capable vehicle routing. Their objective, to sequence orders and minimize the makespan, underscores the importance of streamlined operations and efficient resource utilization. In tandem with these endeavors, our research emphasizes optimizing customer service levels. By integrating insights from such studies with JIT methodologies and AI-driven optimization techniques, we strive to address the complexities of Industry 4.0, fostering agile and responsive supply chains.

Aligned with previous research endeavors, Martins et al. [18] investigated an analogous integrated problem, focusing on a hybrid flow-shop configuration for production. Their study highlights makespan optimization using a mixed integer linear programming model and a biased random variable neighborhood descent (BR-VND) algorithm. Such insights contribute to the ongoing discourse on integrated production and distribution planning systems, paving the way for enhanced operational efficiency and resource utilization within contemporary logistics frameworks.

Expanding on previous investigations, such as those conducted by Martins et al. [18], and in line with the research by Hou et al. [16] addressing an integrated problem within a distributed flow shop and multi-depot vehicle routing environment, Hou et

Table 1: Overview of articles related to integrating production planning with the vehicle routing problem and similar to our work. The table includes columns specifying the machine configuration, type and number of vehicles in VRP, authors' names, and checkmarks indicating whether the article contains one or more mathematical models, heuristics, metaheuristics, and artificial intelligence.

Machine Configuration	Vehicle Routing Problem		Author(s)	Math Model	Heuristic	Metaheuristic	IA
	Type	Number					
Flexible flowshop	Scheduling problem only		Zhu et al. [12]	yes	yes	yes	yes
Hybrid Flow Shop	Scheduling problem only		Nahhas et al. [13]	no	no	no	yes
Vehicle Routing Problem only	Homogeneous	Single	Peng et al. ([14]) Nazari et al. ([15])	no	no	no	yes
Flow shop	Homogeneous	Limited	Hou et al. ([16])	yes	no	yes	no
Flow shop	Homogeneous	Single	Ta et al. [17]	yes	yes	no	no
Permutation flow-shop	Homogeneous	Single	Naganoa et al. ([11])	yes	no	yes	no
Hybrid flow-shop	Homogeneous	Single	Martins et al. ([18]), Wang et al. ([19])	yes	no	yes	no
Flexible job-shop	Heterogeneous	Limited	Mohammadi et al. [5]	yes	no	yes	no
Single machine	Homogeneous	Limited	Tamannaei et al. [7], Liu et al. [20]	yes	yes	yes	no
Single machine	Heterogeneous	Limited	Felix et al. [21], Arroyo et al. [22], Zou et al. [23]	yes	yes	yes	no
Identical parrallel machines	Heterogeneous	Limited	Yagmur et al. [4]	yes	no	yes	no
Parallel machines with machine-dependent ready times	Heterogeneous	Limited	Ullrich et al. [3]	yes	no	yes	no
Unrelated parallel machines with machine-dependent ready times	Heterogeneous	Limited	Araujo et al. [24]	no	yes	no	yes
Unrelated parallel machines with machine-dependent ready times	Heterogeneous	Limited	Our Approach	yes	yes	yes	yes

al. present an enhanced brainstorming optimization (EBSO) algorithm to minimize total weighted advances and delays, comparing its results with various other algorithms, including the discrete artificial bee colony (DABC) algorithm [25], iterative greedy algorithm (IG) [26], genetic algorithm (GA) [27], memetic algorithm (MA) [28], and scatter search algorithm (SS) [29]. These approaches collectively contribute to advancing operational efficiency and optimization within contemporary logistics paradigms.

Wang et al. [19] addressed a three-stage hybrid flow shop scheduling problem incorporating product distribution. Their study involves Stage 1 utilizing a system of identical parallel machines with sequence-dependent setup times, Stage 2 employing dedicated machines, and Stage 3 focusing on the multi-trip traveling salesman problem with capable vehicles, all aimed at minimizing the maximum delivery completion time. To tackle this challenge, Wang et al. proposed a mixed integer linear programming model alongside methods based on variable neighborhood search (VNS), a four-layer constructive heuristic method (C HVNS), and a hybrid heuristic method (CONSVNS) that combines the VNS and C HVNS methods. These strategies contribute to optimizing operational efficiency and logistics within complex production and distribution environments.

Our proposed approach leverages several advanced techniques to address the intricate challenges of integrating production planning on unrelated parallel machines with the vehicle routing problem. Firstly, we introduce a robust Mixed Integer Linear Programming Model (MILPM) to provide a solid mathematical foundation for optimization. Additionally, we deploy a Constructive Heuristic and Neighborhood Search Heuristics to explore solution spaces efficiently. To enhance adaptability and efficiency further, we introduce a novel framework integrating machine learning, enabling predictive analysis to determine the most effective Neighborhood Search Heuristics for a given instance based on its unique characteristics. Moreover, we present an innovative RVND metaheuristic alongside a Hybrid Metaheuristic (PPO-VND), which integrates Variable Neighborhood Descent (VND) with a Reinforcement Learning Algorithm, Proximal Policy Optimization (PPO). These advancements offer tangible gains in solution quality and computational efficiency and highlight the significant advantages of incorporating Artificial Intelligence (AI) into logistics optimization, fostering adaptability and resilience in dynamic operational environments.

The potential contributions of this article are significant. First, we introduce a Mixed Integer Linear Programming (MILP) model that solves the integrated problem of production planning on unrelated parallel machines and the routing of capable vehicles. Furthermore, we present an initial solution algorithm and eight neighborhood search heuristics (NSH). The study also presents an innovative framework based on machine learning that can predict the most effective NSH heuristic to solve each instance. Furthermore, a hybrid metaheuristic combines the heuristic Variable Neighborhood Descent (VND) with the reinforcement learning algorithm Proximal Policy Optimization (PPO), significantly improving solution quality and computational efficiency. Finally, detailed comparisons demonstrate the superiority of the proposed AI approaches over the others presented, with the framework standing out better than NSH and the PPO-VND standing out in instances of greater complexity.

These contributions demonstrate the relevance and innovation of optimizing logistics operations.

This paper's organization is as follows: Sections 2 and 3 state the research problem and describe the mathematical model, respectively. Section 4 presents the Constitutive Heuristics, the Neighborhood Search Heuristics, the framework, the Proximal Policy Optimization, the metaheuristic Random Variable Neighborhood Search (RVND), and the hybrid metaheuristic PPO-VND. In Section 5, we present the computational experiments conducted for this research. Finally, Section 6 concludes the study.

2 Problem Definition

In the present work, we address the problem of production planning on Unrelated parallel machine scheduling with setup times and ready times integrated with the vehicle routing problem, a complex and relevant combination in industrial contexts where efficiency in the production and distribution of products is crucial. This problem consists of determining the assignment of a set of jobs to a set of parallel machines and the subsequent routing of the vehicles that will distribute the finished products to customers. Integrating these two steps adds a layer of complexity and amplifies the need for optimized solutions due to the interdependencies between production and distribution. This problem is NP-Hard since it generalizes the production planning problem on Unrelated parallel machine scheduling with setup times and ready times, known as NP-Hard as presented by Lin and Hsieh [30], and the vehicle routing problem, which belongs to the class of NP-hard problems as presented by Lenstra and Kan [31]. Thus, the combination of these two subproblems inherits this characteristic.

A collection of jobs $I = \{1, 2, ..., n\}$ needs to be processed on m distinct parallel machines, denoted as $M = \{1, 2, ..., m\}$, without preemption, Each job j has specific characteristics: a processing time p_{ij} on the machine i, a due date (d_j), a tardiness penalty (w_j), and a size (h_j) indicating the space it occupies in a vehicle. Additionally, there is a machine-dependent setup time s_{ijk} required to prepare for job k immediately after job j. Generally, $s_{ijk} \neq s_{ikj}$ [24].

Once the jobs have been processed on the machines, they are divided into batches for delivery to clients using a fleet of l vehicles, represented as $L = \{1, 2, ..., l\}$. The routing problem is modeled on a graph $G(V, A)$, where $V = \{0, 1, ..., n\}$ represents the set of nodes, and $A = \{(j, k) : 0 \geq j, k \geq n, j \neq k\}$ represents the set of arcs. The node 0 signifies the depot, and each job corresponds to a customer, with the set of customers being $I = \{1, ..., n\}$. Each arc (j, k) has an associated travel time t_{jk} . Additionally, each vehicle $v \in L$ has a capacity q_v that must not be exceeded. The problem assumes single-use vehicles and one visit per customer. The problem aims to determine the jobs processing orders and the vehicles delivery routes to minimize the jobs total weighted tardiness (TWT) [24].

Table 2 presents the parameters for a problem involving three machines ($m = 3$), six jobs ($n = 6$), and four vehicles ($v = 4$), Table 3 presents setup time for example. Figure 1 illustrates a solution for the problem, showing the completion times for each job, delivery times for each job, and vehicle routes. In this specific instance, the completion and delivery times for job I_1 are 21 and 44, respectively; for job I_2, 45

and 62; for job I_3, 39 and 59; for job I_4, 14 and 36; for job I_5, 8 and 58; and finally, for job I_6, 23 and 61.

Table 2: Parameters for an example involving three machines, six jobs, and four vehicles.

Jobs	Processing Time			Job Information			Distances							
	M_1	M_2	M_3	h_j	d_j	w_j	Jobs	I_0	I_1	I_2	I_3	I_4	I_5	I_6
I_0	0	0	0	0	0	0	I_0	0	19	17	20	22	22	16
I_1	36	1	43	24	47	10	I_1	19	0	18	34	41	36	17
I_2	46	20	32	27	62	9	I_2	17	18	0	18	33	39	29
I_3	39	24	37	24	55	3	I_3	20	34	18	0	21	38	36
I_4	41	14	32	20	62	2	I_4	22	41	33	21	0	22	33
I_5	7	5	8	5	62	9	I_5	22	36	39	38	22	0	19
I_6	31	45	9	19	61	16	I_6	16	17	29	36	33	19	0

Vehicle information				
Vehicle	V_1	V_2	V_3	V_4
q_v	58	65	41	72

Table 3: Setup time an example involving three machines, six jobs, and four vehicles.

	Setup Time M_1								Setup Time M_2								Setup Time M_3							
Job	I_0	I_1	I_2	I_3	I_4	I_5	I_6	Job	I_0	I_1	I_2	I_3	I_4	I_5	I_6	Job	I_0	I_1	I_2	I_3	I_4	I_5	I_6	
I_0	0	0	0	0	0	0	0	I_0	0	0	0	0	0	0	0	I_0	0	0	0	0	0	0	0	
I_1	0	0	1	4	5	2	4	I_1	0	0	4	1	4	5	2	I_1	0	0	4	6	2	3	6	
I_2	0	5	0	3	6	4	6	I_2	0	4	0	3	4	1	4	I_2	0	4	0	8	4	3	2	
I_3	0	2	3	0	3	1	3	I_3	0	6	3	0	7	4	4	I_3	0	7	6	0	6	7	3	
I_4	0	4	2	3	0	4	1	I_4	0	6	2	5	0	3	6	I_4	0	2	3	4	0	1	5	
I_5	0	1	2	4	6	0	2	I_5	0	3	1	4	3	0	5	I_5	0	1	4	5	1	0	6	
I_6	0	3	1	2	5	3	0	I_6	0	2	4	3	5	3	0	I_6	0	5	3	6	3	4	0	

The vehicle V_1 starts at time 14 and is responsible for delivering jobs I_4 and I_5, following the route: $I_0 \rightarrow I_4 \rightarrow I_5 \rightarrow I_0$, where, I_0 represents the depot. Vehicle V_2 starts at time 25 and is responsible for delivering jobs I_1 and I_6, following the route: $I_0 \rightarrow I_1 \rightarrow I_6 \rightarrow I_0$. The vehicle V_3 starts at 45 and is responsible for delivering the job I_2, following the route: $I_0 \rightarrow I_2 \rightarrow I_0$. Finally, the vehicle V_4 starts at time 39 and delivers the job I_3, following the route: $I_0 \rightarrow I_3 \rightarrow I_0$. In this example, only job I_3 was delayed by four time units(4); the penalty for each time unit is three (3). Therefore, the total weighted tardiness is twelve (12).

3 Mixed Integer Linear Programming Model

This section proposes a Mixed Integer Linear Programming (MILP) model for the research problem. Table 4 presents the model parameters and the decision variables:

The model also used the following parameters.

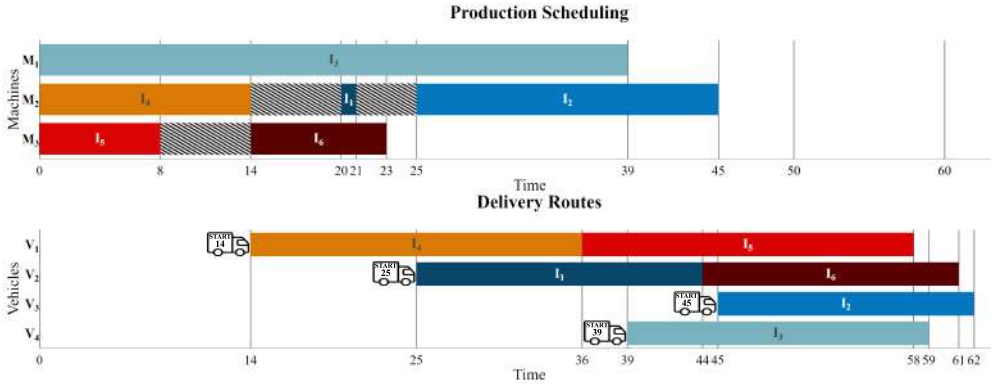


Fig. 1: The scheduling of jobs and delivery routes.

Table 4: Parameters and the decision variables for a Mixed Integer Linear Programming (MILP) model

Indices	
i	Machine.
j, k	Jobs.
v	Vehicle.
O	Origin (Deposit).
Sets	
$M = \{1, 2, ..., m\}$	Set containing m machines.
$I = \{1, 2, ..., n\}$	Set containing n jobs.
$N = I \cup \{O\}$	Set containing jobs and a fictitious job from the source.
$L = \{1, 2, ..., l\}$	Set containing l vehicles.
Parameters	
p_{ij}	Processing time for job j on machine i.
s_{ijk}	Setup time for job k after job j on machine i.
t_{jk}	Travel time between location j and location k.
w_j	Delay penalty for job j.
d_j	Delivery date for job j.
h_j	Customer demand j (job size j).
q_v	Vehicle capacity v.
Decision variables	
Z_{vjk}	1 - If the vehicle v is used to travel between points j and k. 0 - Otherwise.
X_{ijk}	1 - If job j precedes job k on machine i, 0 - Otherwise.
Y_{ij}	1 - If job j is processed on machine i, 0 - Otherwise.
U_v	1 - If the vehicle v is used to make deliveries. 0 - Otherwise.
C_j	Completion time of job j.
S_v	Vehicle trip start time v.
T_j	Delivery delay for job j.
D_j	Delivery time for job j.

- MG is a sufficiently large value defined by $\sum_{i\in M}\sum_{j\in N}\sum_{k\in N} t_{jk} + p_{jk} + s_{ijk}$.
- $N = I \cup \{O\}$, represents a set containing the jobs and a dummy job starting from the origin.

Find below the mathematical model presentation. The objective of the model is:

$$\min \text{ TWT} = \sum_{j\in I} w_j T_j \tag{1}$$

Where,

$$\sum_{v\in L}\sum_{k\in N, k\neq j} Z_{vjk} = 1, \quad \forall j \in I \tag{2}$$

$$\sum_{j\in N}\sum_{k\in N, k\neq j} Z_{vjk} h_j \leq q_v U_v, \quad \forall v \in L \tag{3}$$

$$\sum_{k\in N} Z_{v0k} = U_v, \quad \forall v \in L \tag{4}$$

$$\sum_{j\in N} Z_{vjh} = \sum_{k\in N} Z_{vhk} \forall h \in N, \quad \forall v \in L \tag{5}$$

$$\sum_{k\in I} X_{i0k} \leq 1, \quad \forall i \in M \tag{6}$$

$$\sum_{i\in M} Y_{ij} = 1, \quad \forall j \in I \tag{7}$$

$$Y_{ij} = \sum_{k\in N, k\neq j} X_{ijk}, \quad \forall i \in M, \forall j \in I \tag{8}$$

$$Y_{ik} = \sum_{j\in N, k\neq j} X_{ijk}, \quad \forall i \in M, \forall k \in I \tag{9}$$

$$C_k - C_j + MG(1 - X_{ijk}) \geq s_{ijk} + p_{ik}, \quad \forall j \in N, \forall k \in I, j \neq k, \forall i \in M \tag{10}$$

$$S_v \geq C_k - MG(1 - \sum_{j\in N, j\neq k} Z_{vjk}), \quad \forall v \in L, \forall k \in N \tag{11}$$

$$D_k - S_v \geq t_{0k} - MG(1 - Z_{v0k}), \forall v \in L, \quad \forall k \in N \tag{12}$$

$$D_k - D_j \geq t_{jk} - MG(1 - \sum_{v \in L} Z_{vjk}),$$
$$\forall j \in I, \forall k \in N, j \neq k \tag{13}$$

$$T_j \geq D_j - d_j, \quad \forall j \in I \tag{14}$$

The objective function, defined by Equation 1, is designed to minimize the sum of the weighted job delays. The conditions outlined in 2 ensure that each customer is served exactly once and by a single vehicle. The constraints 3 ensure that vehicle capacities are maintained, with the consideration of $h_0 = 0$. Conditions 4 and 5 ensure that, when utilized, a vehicle must both start and end its route at the depot. Constraints 6 ensure that only one job runs after each machine's dummy job I_0. Constraints 7 guarantee that each job runs on only one machine. Constraints 8 and 9 dictate that each job has precisely one successor and one predecessor. Constraints 10 define the completion time for each job, considering $C_0 = 0$. Constraints 11 specify the starting time for each travel. The constraints 12 and 13 determine vehicle arrival times. Lastly, constraints 14 establish job delays based on their due dates.

4 Proposed Algorithms

In this study, we present two solution-decoding algorithms along with a constructive algorithm. We also introduce neighborhood search heuristics. To improve problem-solving efficiency, we develop a framework that utilizes machine learning to choose the most suitable neighborhood search heuristic for a specific problem instance based on its characteristics.

Furthermore, it presents the Proximal Policy Optimization (PPO) and Reinforcement Learning. Subsequently, the PPO-VND Hybrid Metaheuristic integrates the Variable Neighborhood Algorithm (VND) with the Reinforcement Learning Algorithm PPO proposes. The following section provides detailed explanations of these algorithms.

4.1 Decoding Algorithms

Although the problem has two components (parallel machine scheduling and vehicle routing), the solution is divided into two parts ($sol = \{s, r\}$). Two job lists are used to construct a solution (sol) for the problem: $L1$ and $L2$. The $L1$ list represents the order in which the jobs will be inserted into the machines by the NEH (Nawaz-Enscore-Ham) decoding algorithm. At the end of the execution of the NEH algorithm, a job schedule on the machines (s) is obtained. The $L2$ list, in turn, represents the order in which jobs are inserted into the routes by the PIFH (Push Forward Insertion Heuristic) decoding algorithm. At the end of the execution of the PIFH algorithm, it is possible to obtain the job delivery routes (r). At the end of the decoding algorithms, it is possible to obtain a solution to the problem ($sol = \{s, r\}$). Figure 2 shows the solution decoding process.

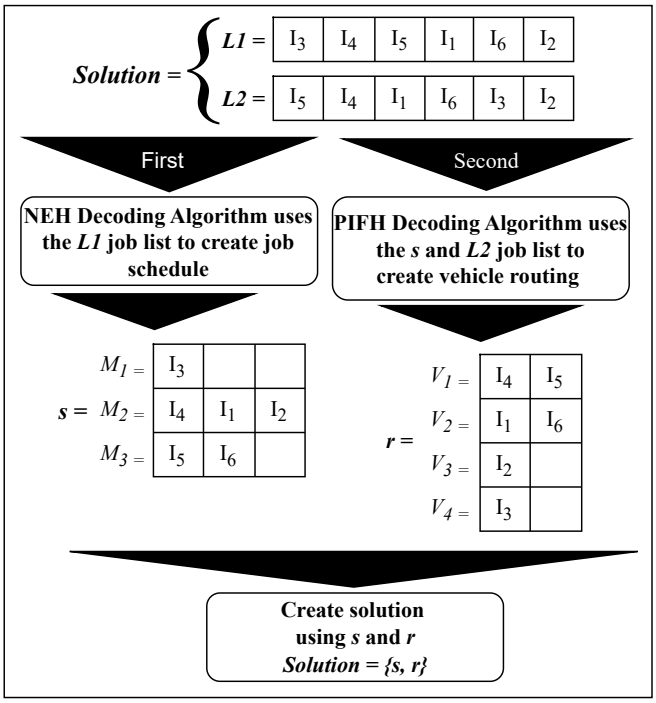


Fig. 2: The solution decoding process.

The NEH heuristic proposed by Nawaz et al. [32] is a greedy algorithm frequently employed in scheduling problems. The decoding algorithm NEH is based on the NEH heuristic. The decoding algorithm NEH inserts jobs from the $L1$ list sequentially on the machines. Jobs are inserted into the schedule in a position that minimizes the job completion time on the machine. After all jobs are inserted into the schedule, the scheduling s is finally defined. The NEH decoding algorithm is presented in Algorithm 1.

Algorithm 1 NEH decoding algorithm.

```
1: function NEH(L1)
2:     s ← ∅                                   ▷ The job scheduling
3:     for i = 1 to n do
4:         Inserts job L1[i] in the best position in the sequencing s
5:     end for
6:     return s
7: end function
```

Solomon's PIFH heuristic [33] is an efficient algorithm for constructing routes. It uses a greedy approach to sequentially insert jobs into routes. The PIFH decoding algorithm places jobs into routes based on the $L2$ list. In each iteration, a job is inserted in a way that minimizes the total weighted tardiness (wt). After inserting all jobs in the routes, the vehicle routes r are finally defined. The PIFH decoding algorithm is presented in Algorithm 2.

Algorithm 2 PIFH decoding algorithm.

```
1: function PIFH( L2, s )
2:     r ← ∅                                          ▷ Vehicle routes
3:     for i = 1 to n do
4:         Inserts job L2[i] in the best position in the route's r
5:     end for
6:     return r
7: end function
```

4.2 Initial Solution

To create a solution to the problem ($sol = \{s, r\}$), it is initially necessary to define the job list $L1$. The job list $L1$ is constructed by ordering the jobs in descending order of the sum of the average processing and setup times. Once $L1$ is constructed, the decoding algorithm NEH is used to define the sequencing of jobs on the machines (s). Subsequently, from the sequence s, the job list, $L2$, is created. The earliest due date (EDD) rule is used to construct the L2 job list, sorting jobs in ascending order based on the difference between the due date and the job completion time ($d_j - C_j$). Once the $L2$ list is constructed, the decoding algorithm PIFH creates the job's delivery routes (r). Figure 3 the process of building the initial solution and the Algorithm 3 presents the creation of the initial solution to the problem.

Algorithm 3 Initial Solution.

```
1: function Initial Solution( )
2:     Create job list L1
3:     s ← NEH(L1)
4:     Create job list L2 from scheduling of jobs on the machine's s
5:     r ← PIFH(L2, s)
6:     solution ← {s, r}
7:     return solution
8: end function
```

4.3 Neighborhood Search Heuristics

In the beginning, we created two algorithms: the Swap Algorithm (Algorithm 4) and the Insertion Algorithm (Algorithm 5). The Swap Algorithm exchanges two position

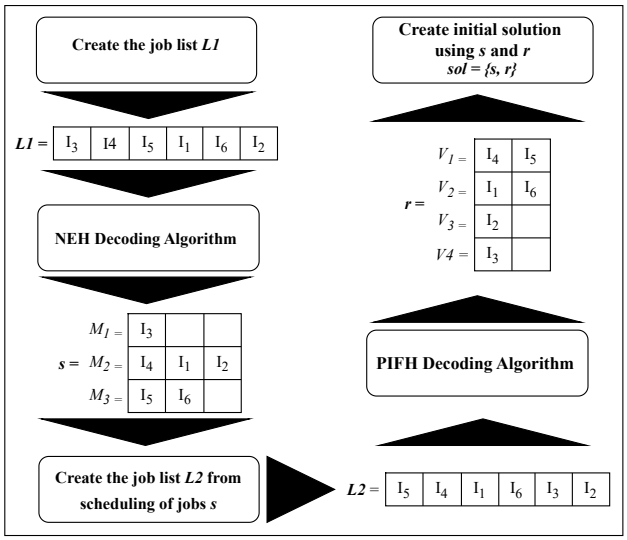


Fig. 3: Initial solution construction process.

list elements to generate a new list. In contrast, the Insertion Algorithm involves removing an element from the list and reinserting it in a different position to create a new list.

Algorithm 4 Swap Algorithm.

```
1: function SWAP( list, i, j )
2:     aux ← list[i]
3:     list[i] ← list[j]
4:     list[j] ← aux
5:     return list
6: end function
```

Algorithm 5 Insertion Algorithm.

```
1: function INSERTION( list, i, j )
2:     aux ← list[i]
3:     list.remove(aux)
4:     list.insert(j, aux)
5:     return list
6: end function
```

Subsequently, we developed the Neighborhood Search Heuristics (NSH, Algorithm 6) to elevate the quality of the solutions. The NSH operates iteratively, testing various swap or insertion movements in the job list ($L1$ or $L2$) and evaluating the solutions generated. Ultimately, the algorithm returns the best solution found during its execution.

The algorithm takes as input two lists of jobs ($L1$ and $L2$) and a set of parameters to determine algorithm behavior. The *movement's* parameter determines whether the algorithm will use exchange or insertion movements, while the *list* parameter specifies the jobs list ($L1$ or $L2$) in which these movements will be executed. Finally, the *restart* parameter determines whether the search should restart if a better solution is discovered during the algorithm's execution. From the parameters, we can define eight Neighborhood Search Heuristics; the identification of variations and the values of each parameter are presented in Table 5.

Table 5: Name of Neighborhood Search Heuristics and Parameters.

Name	Parameters		
	Movements	List	Restart
NSH 1	Swap	L2	False
NSH 2	Swap	L2	True
NSH 3	Insertion	L2	False
NSH 4	Insertion	L2	True
NSH 5	Swap	L1	False
NSH 6	Swap	L1	True
NSH 7	Insertion	L1	False
NSH 8	Insertion	L1	True

The Algorithm 6 begins by creating the initial solution from the $L1$ and $L2$ job lists. The parameters *list* and *movements* determine the list and movement to be used by the algorithm. If the exchange or insertion is applied to the job list $L1$, the decoding algorithm NEH generates a new sequencing of jobs on machines s', which is used to generate a new $L2'$ job list. The PIFH algorithm then uses $L2'$ to define the job delivery routes (r'). In the end, s' and r' create a new solution. If the exchange or insertion is applied to the job list $L2$, the initial sequencing of jobs on the machines is maintained, and the delivery routes define the PIFH algorithm. In the end, s and r' create a new solution. Each solution is compared to the current solution, and if it is superior, it is updated. Finally, the *restart* parameter determines whether to reset the search. In the end, the algorithm returned the best solution found.

4.4 Framework

To effectively tackle combinatorial optimization problems, identifying the algorithm that offers the most consistently favorable outcomes for a given set of instances is often necessary. Traditionally, this approach can deliver impressive results in certain

Algorithm 6 Neighborhood Search Heuristics.

```
1: function NSH( L1, L2, movements, list, restart )
2:     The decoders determine the solution = {s, r} from jobs lists L1 and L2.
3:     best_L1 ← L1
4:     best_L2 ← L2
5:     if list =′ 1′ then ▷ Value '1' indicates that the algorithm uses the L1 Job List.
6:         L ← L1
7:     else
8:         L ← L2
9:     end if
10:    i ← 0
11:    while i ≤ len(L) do
12:        Best_FO =F(solution)
13:        j ← i + 1
14:        while j ≤ len(L) do
15:            if movements = SWAP then
16:                L′ ← swap(L, i, j)
17:            else
18:                L′ ← insertion(L, i, j)
19:            end if
20:            if list =′ 1′ then          ▷ Value '1' indicates that L1 Job List is used.
21:                s′ ← NEH (L′)
22:                Create job list L2′ from scheduling of jobs on the machines s′
23:                r′ ← PIFH (s′, L2′)
24:                solution′ ← {s′, r′}
25:            else
26:                r′ ← PIFH (s, L′)
27:                solution′ ← {s, r′}
28:            end if
29:            if F(solution′) <F(solution) then
30:                solution ← solution′
31:                if list =′ L1′ then
32:                    Best_L1 ← L
33:                    Best_L2 ← L2′
34:                else
35:                    Best_L1 ← L1
36:                    Best_L2 ← L
37:                end if
38:            end if
39:            j+ = 1
40:        end while
41:        i+ = 1
42:        if restart =TRUE AND i = len(L) AND Best_FO ≠F(solution) then
43:            L1 ← Best_L1
44:            L2 ← Best_L2
45:            i ← 0
46:        end if
47:    end while
48:    return solution, Best_L1, Best_L2
49: end function
```

instances, but may falter in others. To address this issue, we developed a novel framework that leverages machine learning (ML) to pinpoint the most effective heuristic for solving a particular problem instance.

Initially, a clustering algorithm is used to group instances with similar characteristics; in this work, we use the k-means algorithm. After clustering, the algorithm created 15 distinct categories of instances. For each instance of the training set, the eight Neighborhood Search Heuristics were executed. After executing the heuristics, the heuristic that produced the best solution was selected to represent that group.

After examining all the patterns and rules in the training set, a machine-learning model is developed to identify the best heuristic for solving a specific instance based on its characteristics. This model considers various factors of the instance, such as the number of jobs (n), number of machines (m), average processing time defined by the equation 15, average setup time defined by the equation 16, average travel time defined by the equation 17, average demand defined by the equation 18, and average vehicle capacity defined by the equation 19. Using these inputs, the ML model predicts the most suitable Neighborhood Search Heuristic (NSH) for addressing a given instance. The goal of training the machine learning model is to develop a function $h(x) : x \to Y$ that accurately predicts the corresponding value of Y for a given x. After training, the model produces a predictor $h(x)$, which, based on the characteristics of a given instance, identifies the most effective Neighborhood Search Heuristic for that instance [24]. The framework is illustrated in Figure 4.

$$\overline{p} = \frac{\sum_{i \in M, j \in I} p_{ij}}{m * n} \tag{15}$$

$$\overline{s} = \frac{\sum_{i \in M, j \in I, k \in I, j \neq k} s_{ijk}}{(m * n^2) - (m * n)} \tag{16}$$

$$\overline{t} = \frac{\sum_{j \in N, k \in N, j \neq k} t_{jk}}{(n+1)^2 - (n+1)} \tag{17}$$

$$\overline{h} = \frac{\sum_{j \in I} h_j}{n} \tag{18}$$

$$\overline{q} = \frac{\sum_{v \in L} q_v}{l} \tag{19}$$

The predictor is a Multilayer Perceptron (MLP) neural network comprising sequential linear layers, where each layer calculates its output through a linear combination of its inputs. The MLP was trained using the Adam algorithm, as outlined in Kingma and Ba [34]. This algorithm is a first-order gradient-based optimization method for stochastic objective functions that relies on adaptive estimation of lower-order moments. Network performance is assessed using the Mean Squared Error (MSE) between the target output and the network's output [24]. Training the neural network took approximately four hours. However, once trained, the model can recommend the optimal algorithm for any instance in about 0.1 seconds.

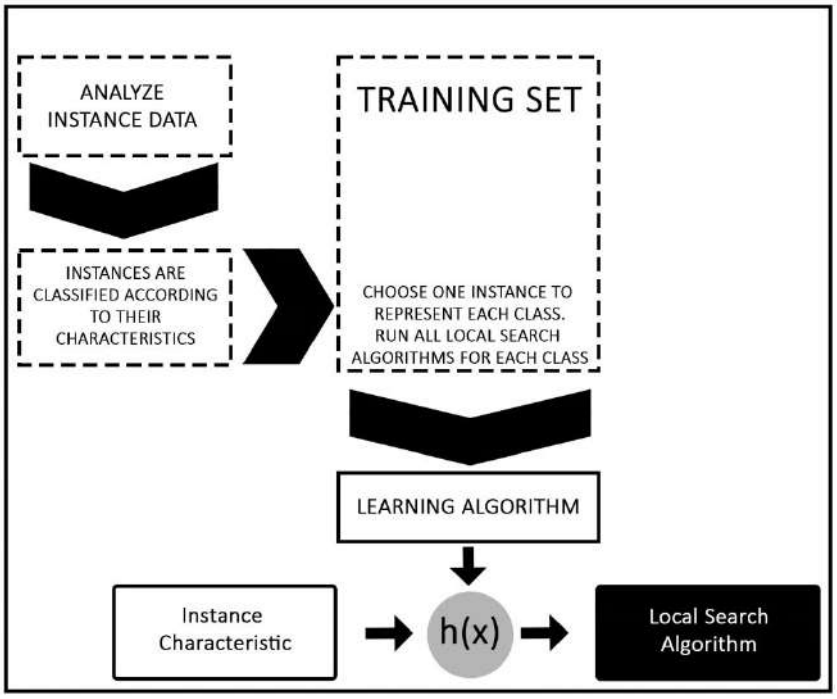


Fig. 4: The Framework Scheme [24].

4.5 The Proximal Policy Optimization

Sequential decision-making problems are often tackled using Reinforcement Learning (RL), which involves an agent interacting with a stochastic environment and learning to map states to actions to maximize cumulative rewards. To model the RL problem, we can use a Markov Decision Process (MDP), which is a tuple $\{S, A, p, r(s, a)\}$. Here, S represents the set of states, A represents the set of actions, and p is the transition function that denotes the conditional probabilities of state change (between 0 and 1). In addition, the reward function $r(s, a)$ assigns a numerical value to the state-action pair. The agent's main objective in the process is to find an optimal approach that associates each state with an optimal action, enabling it to maximize the sum of rewards over time [35].

Schulman et al. [36] introduced the Proximal Policy Optimization (PPO) algorithm, which is based on the Policy Gradients (PG) concept and aims to optimize agents' policies via an on-policy approach. PPO employs a Deep Neural Network (DNN) to map actions to states, enabling DRL agents to learn the best policy directly while maximizing the cumulative reward in the environment. Unlike other methods that learn the function value and derive the best policy, PPO directly finds the best policy.

Actor-critical methods in GP approximate the policy and value function, where the actor learns the policy, and the critic learns the value function to evaluate the stated

policy. Minimizing variation during training speeds up the learning process. Proximal Policy Optimization (PPO) balances implementation, parameterization, and sample complexity. Its primary objective is to limit step size updates, allowing for revamping the policy with one similar to the current one. This process is crucial because extensive updates can cause significant variation in the learning process, leading to poor performance [35]. We utilize the PPO algorithm from the stable-baselines3 library in Python [37]. Algorithm 7 presents the pseudocode for PPO.

Algorithm 7 Proximal Policy Optimization

Initialize Θ and Φ ▷ policy and value-function parameters, respectively
for $i = 0, 1, 2, ...$ **do**
  for $actor = 1, 2, ..., N_{actors}$ **do**
    Run policy $\pi(\Theta)$ for T steps and collect the trajectories.
    Calculate advantage estimates $\hat{A}_1, ..., \hat{A}_T$ based on value-function V_Φ.
  end for
  Form a *batch*, of size NT , with collected trajectories and advantages.
  for $epoch = 1, 2, ..., K$ **do**
    Shuffle *batch* and split into *minibatches*
    for all *minibatches* **do**
      Update Θ wrt objective function via stochastic gradient.
      Update Φ wrt mean-squared error of value-function V_Φ.
    end for
  end for
end for

4.5.1 State, Action, and Reward Function

The state is the data that fully represents the current environment that the agent interacts with. The state's portrayal is a vector feature that describes the environment at a given time: a solution to the problem containing the number of machines (m), the number of jobs (n), the list of the jobs' completion times ($C_j, \forall j \in N$), the list of jobs' delivery dates ($D_j, \forall j \in N$), and the list of tardiness penalties ($w_j T_j, \forall j \in N$).

The action is the decision taken by the agent in a given state and represents the agent's next choice. In this case, the choice is the next neighborhood explored by the local search. The model output is an optimal policy of solution created from job lists $L1$ and $L2$ after the action taken by the agent.

The reward function comes from the objective function, calculated using the Equation (20) and considering the lower bound (LB) and upper bound (UB). LB is calculated using the Equations (21). To calculate UB, we assume that the jobs' distribution will be identical among machines and vehicles. We also consider that the job j will be the last executed on the machine and delivered on the route. Furthermore, all jobs have the longest processing, setup, and travel times possible.

$$\text{Reward Function} = -1 * \frac{(\sum_{j \in I} w_j T_j) - LB)}{(UB - LB)} * 100 \tag{20}$$

$$LB = \sum_{j \in I} w_j * Max((Min(p_{ij} \forall i \in M) \quad + t_{0j} - d_j), 0).$$

4.5.2 The Environment Model

An agent's interaction with the environment requires an environmental model representing the problem. The environment represents the integrated problem of production and distribution planning. The input to the model is two job lists ($L1$ and $L2$) used to create the solution to the problem ($solution = \{s, r\}$). The model's output is the value of the objective function of the solution. The agent's goal is to find a policy that maximizes cumulative rewards over time, which leads to better solutions in the context of the integrated problem. It was necessary to create an environmental model for each group of instances with n equals 10, 15, and 30, as the environmental model is directly linked to the number of jobs. The NEH heuristic inserts the jobs from the $L1$ list sequentially into the solution. At each iteration of the algorithm, a job of $j \in L1$ is inserted in the sequencing of the machines to minimize the machine's completion time. After, the sequencing of the machines s is finally defined. The PIFH heuristic inserts the jobs from the $L2$ job list sequentially to minimize the job's weighted delay (wt). At the end of the algorithm, the vehicle routes r are finally defined. The Environment Model is present in Algorithm 8.

Algorithm 8 Environment Model

1: $solution$ = New solution$\{s, r\}$.
2: Sort the job sequence as $L1$ in descending order of the sum of average processing and setup times.
3: $s \leftarrow$NEH($L1$)
4: Create job list $L2$ from scheduling of jobs on the machines s (EDD rule)
5: $r \leftarrow$ PIFH $(L2, s)$
6: Create $solution$ with s and r
7: Return $f(solution)$

4.6 PPO - VND

The Variable Neighborhood Descent (VND) is a deterministic local search algorithm [38] that employs a series of k neighborhood structures $V = V_1, V_2, ..., V_k$ until the solution S becomes locally optimal. The PPO-VND hybrid metaheuristic is presented in the Algorithm 9. In this algorithm, PPO predicts and defines the Neighborhood Search Heuristics at each PPO-VND iteration. The PPO-VND Scheme is introduced in Figure 5.

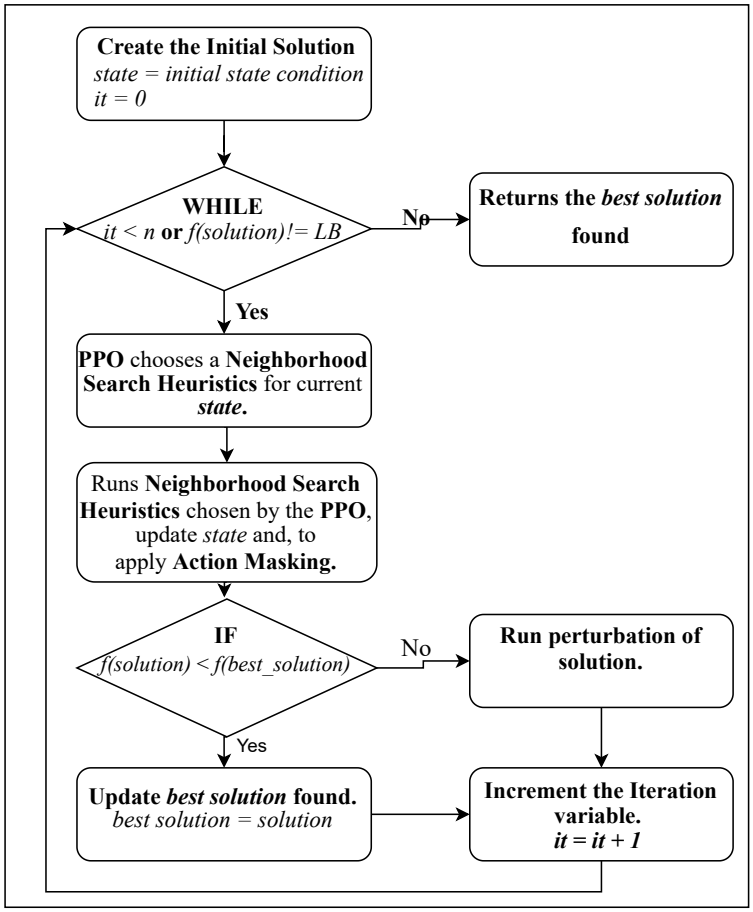


Fig. 5: The PPO-VND Scheme.

The algorithm starts with an initial solution, and at each iteration, the PPO chooses a Neighborhood Search Heuristics. After executing the NSH, a local optimal solution returns. The state is updated, and then action masking is applied. If the new solution is better than the best solution, the algorithm updates it. Otherwise, the algorithm upsets the solution. The process repeats until the number of iterations equals n (number of instance jobs) or the solution found equals the Lower Bound (LB).

The solution's perturbation consists of randomly exchanging and inserting movements in the job lists. The perturbation of the solution is essential in these cases because it prevents the PPO-VND from getting stuck in a local optimal solution, allowing it to explore a more extensive search space. The perturbation happens into 30% of the jobs' list ($L1$ or $L2$, the list chosen randomly), i.e., 30% of jobs are removed

and inserted randomly, creating a new list. Following, the algorithms NEH and PIFH use new job lists ($L1$ and $L2$) to define a new solution, which is later employed in the algorithm. The disturbance parameter was set at 30% based on the calibration process and the systematic analysis using Optuna library [39]. This choice has been shown to provide an ideal balance between exploration and exploitation, allowing algorithms to achieve consistent improvements in the obtained solutions without compromising computational efficiency.

Algorithm 9 PPO - VND

```
1: solution ← INITIALSOLUTION()
2: best_solution ← solution
3: state ← initial state condition
4: it ← 0
5: while it < n or f(solution)! = LB do
6:     u ← PPO.predict(state)              ▷ PPO chooses a NSH for current state.
7:     solution ← NSH u (solution)
8:     state ← update
9:     if F(solution) < F(best_solution) then
10:        best_solution ← solution
11:    else
12:        perturbation of the current solution
13:    end if
14:    it ← it + 1
15: end while
16: return best_solution
```

5 Computational Experiments

This section presents the results of experiments conducted to evaluate the performance of the proposed methods. The Mixed-Integer Linear Programming Model (MILP) was implemented in C++ and solved using the CPLEX solver [40]. The other methods were programmed in Python 3. All methods are executed on an Intel Core i7-4790K (4.0 GHz) CPU with 32 GB of RAM.

We employed the 540 instances proposed by Araujo et al. [24] to conduct the computational experiments. Each problem instance is characterized by two key parameters: the number of jobs denoted as $n \in \{10, 15, 30\}$, and the number of machines represented by $m \in \{2, 4, 8\}$. For example, the instances "N_10_M_2" have 10 jobs and 2 machines.

We trained the PPO agent for 500 thousand episodes with 36 randomly selected instances of each size and different complexities. The PPO hyperparameters were tuned using the RL Baselines3 Zoo library [41].

The results by PPO-VND are compared with the Random Variable Neighborhood Search (RVND) metaheuristic. The RVND is a stochastic local search algorithm [38]

commonly used to solve production planning [42, 43] and vehicle routing problems [44, 45]. Furthermore, we can find works that use RVND to solve integrated production and distribution planning problems [22]. The RVND works similarly to PPO-VND; the only difference between the two algorithms is that in RNVD, the Neighborhood Search Heuristics are chosen randomly.

The algorithms presented in this paper are available at https://github.com/matheusinhofreitas. The computational experiments occurred in a single thread. The processing time for the MILP model is set at a fixed duration of 600 seconds.

To evaluate the quality of the solutions, we use the relative percentage deviation (GAP) of the best-known solution obtained among all methods. The GAP is defined by the Equation 21:

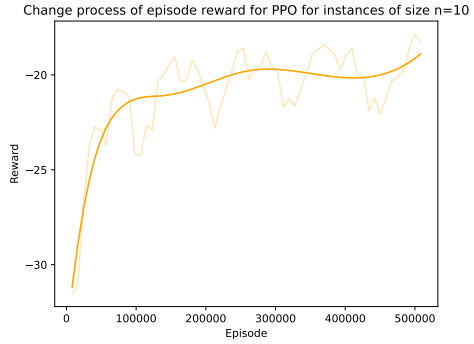
$$GAP = 100 \times \frac{(F_{method} - F_{best})}{F_{method}} \tag{21}$$

Where F_{best} represents the minimum TWT (total weighted tardiness of jobs delays) value obtained among all compared methods, and F_{method} is the TWT obtained with a specific method.

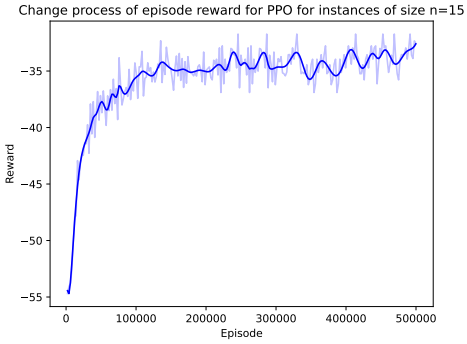
5.1 Computational Results

Initially, we compare the MILP model with PPO-VND, RVND, Framework, and the eight combinations of Neighborhood Search Heuristics (NSH 1, NSH 2, NSH 3, NSH 4, NSH 5, NSH 6, NSH 7, and NSH 8). For each instance, the PPO-VND and RVND algorithms were run five times; as the other methods are deterministic, they were only executed once.

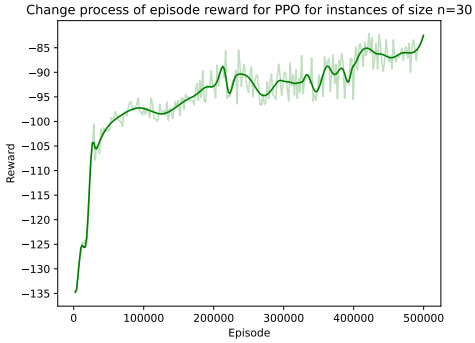
The Figure 6 depicts a steady rise in the average reward during PPO training, signifying the DRL agents' effective learning process in tackling the chosen problem. The orange graph represents the learning curve for instances of size $n = 10$ (figure 6a). The figure 6a shows that the PPO algorithm achieves faster learning for smaller instances compared to larger instances, stabilizing the reward from the first 100 thousand episodes onwards. The blue and green graphs (figure 6b and figure 6c) represent, respectively, the learning curve for instances of size $n = 15$ and $n = 30$. For these instances, it is possible to observe that the PPO has more difficulty carrying out learning, requiring more episodes to stabilize the rewards. In other words, the greater the complexity of the instances, the greater the training time required for the PPO. Even with this difficulty, it is possible to observe in Table 7 the superior performance of PPO-VND compared to other methods. The PPO training time was around 16 hours for instances of size n = 10, around 74 hours for instances of size n = 15, and around 45 days for instances of size n = 30; in our comparisons, we did not consider the training time of the PPO model, since training is performed only once.



(a) Instances size $n = 10$



(b) Instances size $n = 15$



(c) Instances size $n = 30$

Fig. 6: Change process of episode reward for PPO.

We performed an Analysis of Variance (ANOVA) [46] for a 5% significance level. The ANOVA yielded a p-value of 0.00, indicating statistically significant differences between the experiments. Figure 7 shows the boxplot for the GAPs obtained using all methods.

Table 6 presents the average GAPs for the eight Neighborhood Search Heuristics, while Table 7 shows the average GAPs for the other algorithms presented in this work. Each row of these tables represents the average GAPs of the 60 instances, sorted based on the number of jobs and machines. The smallest average GAPs are highlighted in bold. The last row of each table represents the average GAPs for the entire set of instances.

In Table 6, it is possible to observe that the NSH that obtained the best averages was NSH 6, with an average GAP of 37%. Neighborhood Search Heuristic 6 stands out for applying switching movements to the L1 job list and restarting the process whenever a better solution is found. Comparing only the Neighborhood Search Heuristics, NSH 6 managed to find the best values for 386 instances (71%), followed by NSH 8

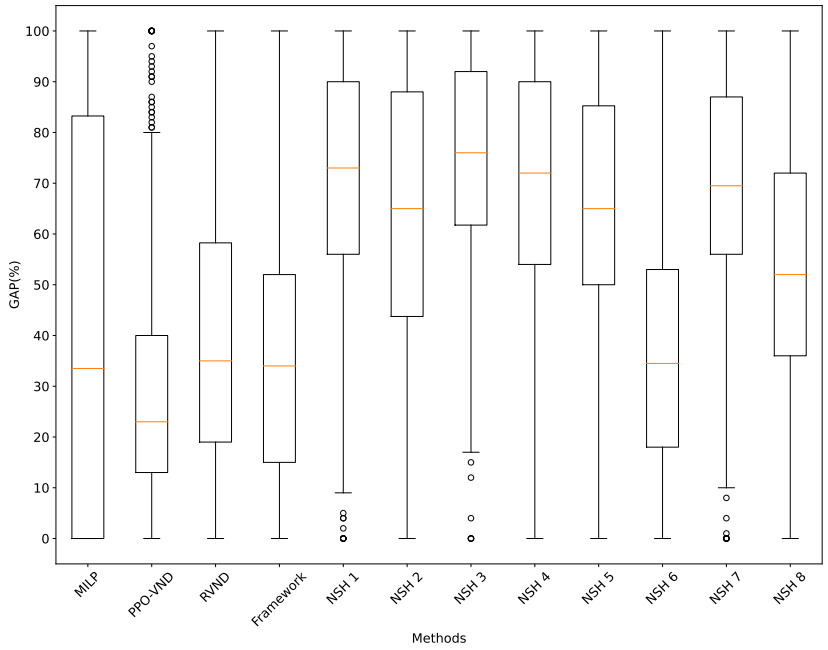


Fig. 7: Boxplot of GAPs (%) for all methods.

with 139 instances (26%). NSH 2 managed to find the best values for 89 instances (16%). NSH 5, NSH 4, NSH 7, NSH 1, and NSH 3 managed to find the best values for 47 (9%), 41 (8%), 37 (7%), 32 (6%), and 26 (5%) instances, respectively.

Table 6: Result for the Neighborhood Search Heuristics

Instances	Average GAP							
	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	83%	81%	85%	83%	78%	**49%**	79%	66%
N_10_M_4	73%	69%	76%	73%	65%	**49%**	69%	56%
N_10_M_8	46%	41%	54%	47%	41%	**29%**	44%	34%
N_15_M_2	88%	85%	89%	87%	81%	**38%**	85%	60%
N_15_M_4	71%	64%	75%	69%	65%	**40%**	69%	50%
N_15_M_8	52%	39%	58%	48%	47%	**30%**	54%	36%
N_30_M_2	90%	87%	91%	90%	89%	**33%**	90%	74%
N_30_M_4	75%	64%	77%	72%	70%	**33%**	72%	56%
N_30_M_8	60%	44%	63%	57%	54%	**30%**	58%	43%
Average	71%	64%	74%	70%	65%	**37%**	69%	53%

Table 7 presents the GAPs achieved by the PPO-VND and RVND algorithms, highlighting their performance across increasingly complex instances. Initially, MILP demonstrates superior average GAPs for smaller instances (n=10). However, as

instance complexity grows, MILP's efficacy diminishes due to the increased job count. In contrast, algorithms employing machine learning, as PPO-VND and the Framework, exhibit improved performance as instance sizes expand.

Table 7: Result average GAP for the presented methods

Instances	Average GAP					Best GAP		Worse GAP	
	MILP	PPO-VND	RVND	Framework	NSH 6	PPO-VND	RVND	PPO-VND	RVND
N_10_M_2	**7%**	41%	59%	49%	49%	34%	45%	46%	70%
N_10_M_4	**5%**	43%	45%	49%	49%	36%	32%	49%	57%
N_10_M_8	**3%**	22%	21%	29%	29%	16%	10%	27%	33%
N_15_M_2	38%	**34%**	65%	36%	38%	22%	51%	40%	75%
N_15_M_4	**21%**	24%	36%	39%	40%	14%	17%	32%	51%
N_15_M_8	37%	26%	**20%**	28%	30%	13%	5%	37%	33%
N_30_M_2	79%	**26%**	69%	31%	33%	4%	56%	43%	77%
N_30_M_4	80%	**24%**	34%	32%	33%	10%	15%	35%	47%
N_30_M_8	90%	**19%**	22%	30%	30%	7%	9%	26%	31%
Average	40%	**29%**	41%	36%	37%	**17%**	27%	**37%**	53%

Additionally, MILP outperforms other methods in small instances, achieving better GAPs in four groups of instances. Specifically, MILP achieves an average GAP of 7% for instances of size (n_10_m_2), 5% for instances of size (n_10_m_4), 3% for instances of size (n_10_m_8), and 21% for instances of size (n_15_m_4). Moreover, MILP identifies optimal solutions for 34 instances, which corresponds to 6% of all instances.

Conversely, the PPO-VND hybrid algorithm also showcased its prowess, securing the best average GAP for four groups of instances. Its adaptability was particularly evident in more complex instances, where it achieved the best GAPs for the groups n_15_m_2 (34%), n_30_m_2 (26%), n_30_m_4 (24%), and n_30_m_8 (19%). This adaptability sparks interest in its potential to handle complex optimization scenarios.

Analyzing the data in Tables 6 and 7, it's evident that both NSH 6 and Framework outperform the RVND meta-heuristic across four instance groups. This underscores the significance of selecting a proficient neighborhood over random selection. Consequently, the findings underscore the pivotal role of intelligent neighborhood selection in the efficacy of these methodologies and their potential to surpass conventional approaches.

Interestingly, while NSH 6 emerged as the most effective Neighborhood Search Heuristic algorithm across all datasets, the framework's performance rivaled or even surpassed NSH 6 in some cases. This proves that using an algorithm that uses machine learning to determine the best heuristic for a specific instance is efficient. The framework's success rate in this experiment was 72%; in other words, for 391 instances (out of a total of 540), the framework chose the best NSH to resolve them. The results found by the framework were good and, in some cases, better than the RVND meta-heuristic.

The Table 8 presents algorithm execution times across various problem instances, unveiling discernible performance trends. MILP consistently exhibits lengthier execution times compared to other algorithms, implying its elevated computational complexity. Conversely, both the PPO-VND and RVND algorithms demonstrate only marginal increases in execution times as the problem size expands, suggesting modest

scalability. Furthermore, the Framework algorithm displays nearly negligible growth in execution times with increasing problem size.

Table 8: Algorithm execution time in seconds.

Instance	MILP	PPO-VND	RVND	Framework	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	564.70	1.46	1.53	0.57	0.16	0.27	0.30	0.72	0.15	0.29	0.28	0.72
N_10_M_4	547.06	1.21	1.85	0.44	0.19	0.30	0.37	0.84	0.18	0.35	0.34	0.87
N_10_M_8	528.21	2.02	2.38	0.56	0.26	0.39	0.51	1.03	0.26	0.45	0.48	1.23
N_15_M_2	593.68	7.40	11.61	0.90	0.71	1.06	1.38	3.91	0.70	1.24	1.28	3.56
N_15_M_4	599.71	10.12	13.76	1.04	0.83	1.38	1.62	4.83	0.82	1.56	1.53	4.51
N_15_M_8	598.07	17.61	17.68	1.54	1.13	1.83	2.23	5.06	1.13	2.05	2.09	6.23
N_30_M_2	600.57	146.83	300.57	12.96	10.62	16.55	20.61	82.44	10.54	18.23	19.85	63.48
N_30_M_4	600.68	167.22	354.30	15.78	12.29	19.96	24.20	100.94	12.38	21.55	23.28	69.80
N_30_M_8	601.20	269.30	469.47	19.75	16.20	24.62	32.17	113.66	17.78	27.54	30.66	88.10
Average	581.54	69.24	130.35	5.95	4.71	7.37	9.27	34.83	4.88	8.14	8.87	26.50

Analysis of all instances reveals that PPO-VND is the leading method, with a GAP of 29%. The framework follows, achieving a GAP of 36%. NSH 6 ranks third with a GAP of 37%. The remaining algorithms are classified as follows: MILP (40%), RVND (41%), NSH 8 (53%), NSH 2 (64%), NSH 5 (65%), NSH 7 (69%), NSH 4 (70%), NSH 1 (71%), and finally, NSH 3 (74%).

To evaluate the percentage of improvement caused by the heuristic relative to the initial solution, the Relative Percentage Improvement (RPI) was calculated. The RPI metric is defined by the equation 22:

$$RPI(\%) = 100 \times \frac{(F_{s.i.} - F_{method})}{F_{s.i.}} \tag{22}$$

Where $F_{s.i.}$ is the TWT of the initial solution provided by the constructive heuristic, and F_{method} the TWT obtained with a specific method.

Table 9 presents the RPI(%) values for all evaluated methods. The results show that the evaluated methods, PPO-VND, RVND, and Framework, achieved significant improvements compared to the initial solution. The PPO-VND method found the best solutions for the groups N_10_M_2 (80%), N_10_M_4 (70%), N_15_M_2 (85%), N_15_M_4 (71%), N_30_M_2 (89%), N_30_M_4 (73%), and N_30_M_8 (60%), followed by RVND, which found the best values for the groups N_15_M_8 (59%) and N_15_M_8 (62%). Overall, PPO-VND had the highest average RPI (71%), followed by RVND with 69%, Framework with 67%, and NSH. The results of PPO-VND highlight its consistent ability to significantly improve initial solutions, especially in more complex instances.

6 Conclusion

This study introduces a range of innovative methodologies aimed at tackling integrated production programming and distribution problems. Among these are a mixed integer linear programming model, the PPO-VND hybrid heuristic, eight neighborhood

Table 9: Result average RPI for the presented methods

Instance	Relative Percentage Improvement (RPI)										
	PPO-VND	RVND	Framework	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	**80%**	74%	76%	31%	36%	21%	30%	53%	76%	48%	67%
N_10_M_4	**70%**	69%	63%	35%	42%	25%	35%	51%	65%	43%	58%
N_10_M_8	57%	**59%**	51%	40%	45%	28%	37%	45%	53%	40%	49%
N_15_M_2	**85%**	78%	83%	27%	38%	18%	29%	55%	84%	46%	75%
N_15_M_4	**71%**	68%	65%	27%	39%	17%	29%	43%	65%	33%	57%
N_15_M_8	59%	**62%**	56%	36%	49%	26%	38%	43%	56%	35%	51%
N_30_M_2	**89%**	80%	87%	17%	36%	8%	22%	32%	87%	26%	70%
N_30_M_4	**73%**	70%	68%	17%	40%	9%	23%	35%	68%	29%	53%
N_30_M_8	**60%**	59%	54%	20%	43%	13%	25%	31%	54%	24%	44%
Average	**71%**	69%	67%	28%	41%	18%	30%	43%	67%	36%	58%

search heuristics, and a framework leveraging machine learning to select the optimal Neighborhood Search Heuristic (NSH) for a given instance. Through the application of the Proximal Policy Optimization (PPO) Reinforcement Learning (RL) algorithm to anticipate the NSH for each PPO-VND interaction, the research presents a dynamic and adaptive approach to problem-solving. Computational experiments demonstrate the superior effectiveness of PPO-VND over traditional mathematical models and alternative algorithms in terms of both execution time and solution quality. Notably, we did not include the training time for PPO and the framework, as it occurred only once.

It's essential to underscore that while MILP exhibited superior performance in small instances characterized by constrained search spaces, its effectiveness diminished as instance complexity increased. This limitation highlights the challenge MILP faces in resolving real-world scenarios with intricate parameters. Conversely, PPO-VND emerged as the preferred approach in such scenarios, consistently outperforming alternative resolution techniques as the number of jobs expanded to 15 and 30. This underscores the adaptability and scalability of PPO-VND in tackling complex instances. Looking ahead, we anticipate further refinement and optimization of the PPO-VND algorithm, particularly in addressing larger instances. Leveraging its capabilities, they aim to confront and overcome the challenges posed by real-world production and distribution problems in future research endeavors.

References

[1] Chen, Z.-L.: Integrated production and distribution operations. Handbook of quantitative supply chain analysis, 711–745 (2004)

[2] Chen, Z.-L.: Integrated production and outbound distribution scheduling: review and extensions. Operations research **58**(1), 130–148 (2010)

[3] Ullrich, C.A.: Integrated machine scheduling and vehicle routing with time windows. European Journal of Operational Research **227**(1), 152–165 (2013) https://doi.org/10.1016/j.ejor.2012.11.049

[4] Yağmur, E., Kesen, S.E.: Integrated production scheduling and vehicle routing problem with energy efficient strategies: Mathematical formulation and meta-heuristic algorithms. Expert Systems with Applications **237**, 121586 (2024) https://doi.org/10.1016/j.eswa.2023.121586

[5] Mohammadi, S., Al-e-Hashem, S.M.J.M., Rekik, Y.: An integrated production scheduling and delivery route planning with multi-purpose machines: A case study from a furniture manufacturing company. International Journal of Production Economics **219**, 347–359 (2020) https://doi.org/10.1016/j.ijpe.2019.05.017

[6] Boudia, M., Louly, M.A.O., Prins, C.: Fast heuristics for a combined production planning and vehicle routing problem. Production Planning and Control **19**(2), 85–96 (2008)

[7] Tamannaei, M., Rasti-Barzoki, M.: Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem. Computers & Industrial Engineering **127**, 643–656 (2019)

[8] Ghinato, P.: Sistema toyota de produção: mais do que simplesmente just-in-time. Production **5**, 169–189 (1995)

[9] Monden, Y.: Toyota Production System: an Integrated Approach to Just-in-time. CRc Press, Boca Raton, FL (2011)

[10] Sugimori, Y., Kusunoki, K., Cho, F., UCHIKAWA, S.: Toyota production system and kanban system materialization of just-in-time and respect-for-human system. The international journal of production research **15**(6), 553–564 (1977)

[11] Nagano, M., Tomazella, C., Tavares-Neto, R., Abreu, L.: Solution methods for the integrated permutation flowshop and vehicle routing problem. Journal of Project Management **7**(3), 155–166 (2022)

[12] Zhu, J., Wang, H., Zhang, T.: A deep reinforcement learning approach to the flexible flowshop scheduling problem with makespan minimization. In: 2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS), pp. 1220–1225 (2020). IEEE

[13] Nahhas, A., Kharitonov, A., Turowski, K.: Deep reinforcement learning techniques for solving hybrid flow shop scheduling problems: Proximal policy optimization (ppo) and asynchronous advantage actor-critic (a3c) (2022)

[14] Peng, B., Wang, J., Zhang, Z.: A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems, 636–650 (2020). Springer

[15] Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. Advances in neural information processing systems **31** (2018)

[16] Hou, Y., Fu, Y., Gao, K., Zhang, H., Sadollah, A.: Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows. Expert Systems with Applications **187**, 115827 (2022)

[17] Ta, Q.C., Billaut, J.-C., Bouquard, J.-L.: Heuristic algorithms to minimize the total tardiness in a flow shop production and outbound distribution scheduling problem. In: 2015 International Conference on Industrial Engineering and Systems Management (IESM), pp. 128–134 (2015). IEEE

[18] Martins, L.d.C., Gonzalez-Neira, E.M., Hatami, S., Juan, A.A., Montoya-Torres, J.R.: Combining production and distribution in supply chains: The hybrid flow-shop vehicle routing problem. Computers & Industrial Engineering **159**, 107486 (2021)

[19] Wang, S., Wu, R., Chu, F., Yu, J.: Variable neighborhood search-based methods for integrated hybrid flow shop scheduling with distribution. Soft Computing **24**(12), 8917–8936 (2020)

[20] Liu, L., Li, W., Li, K., Zou, X.: A coordinated production and transportation scheduling problem with minimum sum of order delivery times. Journal of Heuristics **26**(1), 33–58 (2020)

[21] Felix, G.P., Arroyo, J.E.C.: HeurÍsticas para o sequenciamento da produÇÃo e roteamento de veÍculos com frota heterogÊnea. LII Simpósio Brasileiro de Pesquisa Operacional (2020)

[22] Félix, G.P., Arroyo, J.E.C., Freitas, M.: Iterated local search heuristic for integrated single machine scheduling and vehicle routing. In: Kumar, S., Sharma, H., Balachandran, K., Kim, J.H., Bansal, J.C. (eds.) Third Congress on Intelligent Systems, pp. 223–235. Springer, Singapore (2023)

[23] Zou, X., Liu, L., Li, K., Li, W.: A coordinated algorithm for integrated production scheduling and vehicle routing problem. International Journal of Production Research **56**(15), 5005–5024 (2018)

[24] Araujo, M.d.F., Arroyo, J.E.C., Nogueira, T.H.: Heuristics assisted by machine learning for the integrated production planning and distribution problem. In: International Conference on Intelligent Systems Design and Applications, pp. 120–131 (2022). Springer

[25] Duan, J.-H., Meng, T., Chen, Q.-D., Pan, Q.-K.: An effective artificial bee colony for distributed lot-streaming flowshop scheduling problem, 795–806 (2018). Springer

[26] Mao, J.-y., Pan, Q.-k., Miao, Z.-h., Gao, L.: An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flow-shop scheduling problem with preventive maintenance. Expert Systems with

Applications **169**, 114495 (2021)

[27] Zhang, X., Li, X.-T., Yin, M.-H.: An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem. International Journal of Bio-Inspired Computation **15**(2), 113–124 (2020)

[28] Kurdi, M.: A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem. Applied Soft Computing **94**, 106458 (2020)

[29] Pan, Q.-K., Gao, L., Wang, L., Liang, J., Li, X.-Y.: Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. Expert Systems with Applications **124**, 309–324 (2019)

[30] Lin, Y.-K., Hsieh, F.-Y.: Unrelated parallel machine scheduling with setup times and ready times. International Journal of Production Research **52**(4), 1200–1214 (2014)

[31] Lenstra, J.K., Kan, A.R.: Complexity of vehicle routing and scheduling problems. Networks **11**(2), 221–227 (1981)

[32] Nawaz, M., Enscore Jr, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega **11**(1), 91–95 (1983)

[33] Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research **35**(2), 254–265 (1987)

[34] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[35] Alves, J.C., Mateus, G.R.: Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. In: International Conference on Computational Logistics, pp. 584–599 (2020). Springer

[36] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[37] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021)

[38] Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European journal of operational research **130**(3), 449–467 (2001)

[39] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)

[40] Cplex, I.I.: V12. 1: User's manual for cplex. International Business Machines Corporation **46**(53), 157 (2009)

[41] Raffin, A.: RL Baselines3 Zoo. GitHub (2020)

[42] Haddad, M.N., Cota, L.P., Souza, M.J.F., Maculan, N.: Aiv: A heuristic algorithm based on iterated local search and variable neighborhood descent for solving the unrelated parallel machine scheduling problem with setup times. In: International Conference on Enterprise Information Systems, vol. 2, pp. 376–383 (2014). SCITEPRESS

[43] Yildiz, S.T., Ozcan, S., Cevik, N.: Variable neighborhood search-based algorithms for the parallel machine capacitated lot-sizing and scheduling problem. Journal of Engineering Research, 100145 (2023)

[44] Subramanian, A., Uchoa, E., Ochi, L.S.: A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research **40**(10), 2519–2531 (2013)

[45] Penna, P.H.V., Subramanian, A., Ochi, L.S.: An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. Journal of Heuristics **19**(2), 201–232 (2013)

[46] Zar, J.H.: Biostatistical analysis. 4th. New Jersey, USA, 929 (1999)