UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Vívian Ludimila Aguiar Santos

Abordagem Multiobjetivo para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não-Relacionadas com a Deterioração da Máquina Dependente da Sequência

Belo Horizonte

2023

Vívian Ludimila Aguiar Santos

Abordagem Multiobjetivo para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não-Relacionadas com a Deterioração da Máquina Dependente da Sequência

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutora em Engenharia Elétrica.

Orientador: Prof. Dr. Frederico Gadelha Guimarães

Coorientadora: Profa. Dra. Miri Weiss Cohen

Belo Horizonte
2023

Santos, Vívian Ludimila Aguiar.
S237a Abordagem multiobjetivo para o problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas com a deterioração da máquina dependente da sequência [recurso eletrônico] / Vívian Ludimila Aguiar Santos. - 2023.
1 recurso online (137 f. : il., color.) : pdf.

Orientador: Frederico Gadelha Guimarães.
Coorientadora: Miri Weiss Cohen.

Tese (doutorado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Apêndices: f. 112-137.

Bibliografia: f. 104-111.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Máquinas - Teses. 3. Otimização multiobjetivo - Teses. 4. Manutenção - Teses. 5. Algoritmos - Teses. 6. Cálculos numéricos - Teses. 7. Heurística - Teses. I. Guimarães, Frederico Gadelha. II. Cohen, Miri Weiss. III. Universidade Federal de Minas Gerais. Escola de Engenharia. IV. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pela bibliotecária Ângela Cristina Silva CRB/6 2361
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

"ABORDAGEM MULTIOBJETIVO PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM A DETERIORAÇÃO DA MÁQUINA DEPENDENTE DA SEQUÊNCIA"

VÍVIAN LUDIMILA AGUIAR SANTOS

Tese de Doutorado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Doutor em Engenharia Elétrica. Aprovada em 27 de novembro de 2023. Por:

Prof. Dr. Frederico Gadelha Guimarães
DCC (UFMG) - Orientador

Prof. Dr. Miri Weiss-Cohen
Software Engineering (Braude College) - Coorientadora

Prof. Dr. Marcone Jamilson Freitas Souza
DECOM (UFOP)

Prof. Dr. José Elias Claudio Arroyo
DPI (UFV)

Prof. Dr. Luciano Perdigão Cota
(ITV)

Prof. Dr. Lucas de Souza Batista
DEE (UFMG)

Documento assinado eletronicamente por **Frederico Gadelha Guimaraes**, **Professor do Magistério Superior**, em 27/11/2023, às 17:04, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Marcone Jamilson Freitas Souza**, **Usuário Externo**, em 29/11/2023, às 17:44, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Lucas de Souza Batista**, **Professor do Magistério Superior**, em 30/11/2023, às 12:39, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Luciano Perdigão Cota**, **Usuário Externo**, em 30/11/2023, às 14:02, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Jose Elias Claudio Arroyo**, **Usuário Externo**, em 05/12/2023, às 14:25, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Miri Giss Weiss Cohen**, **Usuária Externa**, em 06/12/2023, às 14:56, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2838517** e o código CRC **465F95F6**.

Com todo o meu amor e carinho, dedico este trabalho ao meu filho Thácio Gabriel, que é a luz da minha vida, a razão do meu sorriso e o maior presente que Deus me deu.

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida, saúde, força, disposição e fé para superar todos os desafios que enfrentei até aqui. Pela proteção e por todas as bênçãos concedidas.

Ao meu marido Thales, minha profunda gratidão por todo amor, amizade, carinho, paciência, dedicação e disposição a me ajudar. Sem a sua presença e apoio inabaláveis, não apenas teria sido difícil iniciar este trabalho, como também impossível concluí-lo. Ao meu filho Thácio, sua chegada trouxe uma nova dimensão de amor e felicidade à minha vida, me ensinando a ser mais paciente, mais compassiva e a valorizar cada momento. Obrigada, meu filho, você trouxe um novo significado à minha vida e você é minha maior inspiração.

Aos meus pais, Valdemar e Flor, pelo amor, orações, educação, por todo apoio e incentivo aos meus estudos e à minha vida profissional. Vocês são meus exemplos de vida. Aos meus irmãos, Vera e Mateus, e meus respectivos cunhados pelo companheirismo, união e amizade, e aos meus sobrinhos pela alegria e gargalhadas. Agradeço a toda minha família pelo apoio e ótima convivência. Em especial, agradeço à minha tia Denise, Herculano e Bernardo, por nos acolher e abrigar em seu lar em Belo Horizonte. Não temos palavras para agradecer a hospitalidade e a gentileza de vocês conosco. Agradeço também à minha vozinha Maria da Conceição (*in memoriam*), por todos os momentos maravilhosos que compartilhamos juntas, sua presença em minha vida foi um presente inestimável.

Ao meu orientador, Prof. Dr. Frederico G. Guimarães, por me receber no MINDS. Agradeço pela paciência em sua orientação, competência, incentivo, conselhos, dedicação, atenção, apoio e confiança. Muito obrigada por acreditar em mim e me ajudar a alcançar meus objetivos. À minha coorientadora, Prof.ª Dr.ª Miri W. Cohen, em que seus conselhos, sugestões e correções foram fundamentais para o aperfeiçoamento deste trabalho.

Aos meus ex-professores, Dr.ª Luciana Assis e Dr. Alessandro Vivas que sempre me apoiaram e deram conselhos valiosíssimos em minha formação. Agradeço por todo suporte computacional para a realização dos meus experimentos na UFVJM.

Aos meus ex-orientadores, Prof. Dr. José Elias Arroyo e Prof. Dr. André Santos, que no mestrado, despertaram em mim o interesse pelo tema. Agradeço pelos ensinamentos e conhecimentos construídos. Gostaria de agradecer também ao Prof. Dr. Marcone Souza por suas grandes contribuições, e aos membros da banca, Prof. Dr. Lucas Batista e Prof. Dr. Luciano Cota, por suas importantes contribuições e sugestões.

Aos meus amigos, pela companhia, em especial, Elizângela e Jhonny. Aos colegas do MINDS e aos professores do doutorado no PPGEE, pela colaboração e aprendizado.

Por fim, meu muito obrigada ao IFNMG – campus Pirapora, por proporcionar o meu crescimento profissional e acadêmico.

“Quem caminha sozinho pode até chegar mais rápido,
mas aquele que vai acompanhado,
com certeza vai mais longe.”
(Clarice Lispector)

RESUMO

Este trabalho aborda um problema de sequenciamento de máquinas paralelas não-relacionadas, nas quais as tarefas exercem um impacto significativo na deterioração das máquinas. Essa deterioração, por sua vez, tem um efeito prejudicial no desempenho das máquinas, resultando em aumentos progressivos nos tempos de processamento das tarefas ao longo do tempo. A fim de resolver esse desafio, é proposto um modelo de programação não-linear inteiro misto, que visa otimizar dois objetivos simultaneamente: minimizar o tempo máximo de conclusão das tarefas, conhecido como makespan, e minimizar o tempo total de atraso na entrega das tarefas. Uma abordagem inovadora é desenvolvida para estender a meta-heurística *Iterated Local Search* (ILS) para problemas multiobjetivos. O algoritmo resultante, denominado *Iterated Local Search Based on Decomposition* (ILS/D), emprega uma estratégia de decomposição semelhante àquela empregada pelo *Multiobjective Evolutionary Algorithm Based on Decomposition* (MOEA/D). Nesse contexto, o ILS é utilizado como mecanismo de busca para aprimorar o processo de exploração dentro da estrutura do MOEA/D. Uma das vantagens distintivas do ILS/D é que, sob o regime de decomposição e agregação, um ILS mono-objetivo pode ser empregado para otimizar cada subproblema, dispensando assim a necessidade de uma busca local multiobjetivo. Para avaliar a eficácia do ILS/D, foram realizadas comparações com outros algoritmos, incluindo o MOEA/D, o *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) e o *Pareto Iterated Local Search* (PILS). Os resultados obtidos demonstram que o ILS/D supera de maneira significativa os demais algoritmos mencionados. Essas descobertas evidenciam não apenas a eficácia da estratégia de decomposição em algoritmos evolutivos, mas também a viabilidade de estender com sucesso o algoritmo ILS para a resolução de problemas de otimização multiobjetivo complexos. Ademais, é proposta uma abordagem multiobjetivo que envolve operações de manutenção. Nessa abordagem, busca-se a integração de manutenções no sequenciamento da produção, com o objetivo de mitigar a deterioração das máquinas e, consequentemente, reduzir o tempo total de processamento. O propósito central é determinar a alocação estratégica das manutenções, referidas como tarefas de manutenção, de forma a maximizar o desempenho global do sistema. Quando ocorre uma falha em uma máquina dentro de um sistema de produção, ou quando seu nível de deterioração atinge um patamar crítico, essa máquina fica incapaz de continuar a produção até que seja restaurada a um estado plenamente operacional por meio de uma intervenção de manutenção. Em outras palavras, o desempenho da máquina deve ser restabelecido a 100%. O período de inatividade de uma máquina resulta em perdas de tempo de produção e pode sobrecarregar outras máquinas no sistema, levando-as também a ficarem indisponíveis. Nesse contexto, são desenvolvidas três estratégias distintas para o escalonamento das tarefas de manutenção, todas operando dentro do algoritmo ILS/D. Um conjunto abrangente de experimentos numéricos é condu-

zido em instâncias de diversos tamanhos, demonstrando que os algoritmos desenvolvidos têm a capacidade de oferecer soluções mais precisas para o problema de programação da manutenção.

Palavras-chave: deterioração da máquina; deterioração dependente da sequência; sequenciamento da manutenção; otimização multiobjetivo; meta-heurísticas; makespan, atraso total; máquinas paralelas não-relacionadas.

ABSTRACT

This work addresses a scheduling problem unrelated to parallel machines, in which jobs significantly impact machine deterioration. This deterioration, in turn, adversely affects machine performance, resulting in progressive increases in job processing times over time. To tackle this challenge, a mixed-integer nonlinear programming model is proposed, aiming to optimize two objectives simultaneously: minimizing the maximum job completion time, known as makespan, and minimizing the job total tardiness. An innovative approach is developed to extend the meta-heuristic Iterated Local Search (ILS) to multiobjective problems. The resulting algorithm, named Iterated Local Search Based on Decomposition (ILS/D), employs a decomposition strategy similar to that used by the Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D). In this context, ILS is utilized as a search mechanism to enhance the exploration process within the MOEA/D framework. One distinctive advantage of ILS/D is that a single-objective ILS can optimize each subproblem under the decomposition and aggregation framework, thus obviating the need for multiobjective local search. To evaluate the effectiveness of ILS/D, comparisons were made with other algorithms, including MOEA/D, Non-dominated Sorting Genetic Algorithm II (NSGA-II), and Pareto Iterated Local Search (PILS). The results demonstrate that ILS/D significantly outperforms the other mentioned algorithms. These findings highlight the decomposition strategy's effectiveness in evolutionary algorithms and illustrate the ILS algorithm's successful extension to complex multiobjective problem resolution. Furthermore, a multiobjective approach involving maintenance is proposed. In this approach, the integration of maintenance into production scheduling is sought, to mitigate machine deterioration and reduce the total processing time. The central purpose is to determine the strategic allocation of maintenance, or maintenance jobs, to maximize the overall system performance. When a machine fails within a production system or when its level of deterioration reaches a critical threshold, that machine becomes unable to continue production until it is restored to a fully operational state through maintenance intervention. In other words, the machine's performance must be restored to 100%. Machine downtime results in production time losses and can overload other machines in the system, causing them to become unavailable as well. In this context, three distinct strategies for scheduling maintenance jobs are developed, all operating within the ILS/D algorithm. A comprehensive set of numerical experiments is conducted on instances of various sizes, demonstrating that the developed algorithms can provide more precise solutions to the maintenance scheduling problem.

Keywords: machine deterioration; sequence-dependent deterioration; maintenance scheduling; multiobjective optimization; meta-heuristics; makespan; total tardiness; unrelated parallel machines.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANOVA	Análise de Variância
CR	*Congestion Ratio*
HV	Hipervolume
ILS	*Iterated Local Search*
ILS/D	*Iterated Local Search Based on Decomposition*
MINLP	*Mixed Integer Nonlinear Programming*
MOEA/D	*Multiobjective Evolutionary Algorithm Based on Decomposition*
MS1	*Maintenance Strategy 1*
MS2	*Maintenance Strategy 2*
MS3	*Maintenance Strategy 3*
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
PILS	*Pareto Iterated Local Search*
PLS	*Pareto Local Search*
SA	*Simulated Annealing*
RVND	*Random Variable Neighborhood Descent*
TDT	*Total Delay Time*

SUMÁRIO

1 Introdução

A programação da produção busca realizar a atribuição dos recursos para o conjunto de processos de fabricação, de forma eficiente, e determinar o mais adequado tempo para executar cada operação ou tarefa, tendo em conta as relações entre os processos de fabricação e a capacidade limitada dos recursos de produção. Além disso, a programação da produção busca otimizar os critérios que estão associados à utilização eficiente dos recursos, como por exemplo, diminuição no tempo total e custos da produção, atendimento das datas combinadas para entrega dos produtos, e obtenção de qualidade, confiabilidade, velocidade e flexibilidade nos processos industriais (RUSSOMANO, 2000; LUSTOSA; MESQUITA; OLIVEIRA, 2008).

Existem problemas da programação da produção em que os tempos de processamento das tarefas são geralmente conhecidos com antecedência e permanecem constantes durante o sequenciamento. Ou seja, estes problemas de escalonamento possuem tempos fixos de processamento (PINEDO, 2016). No entanto, essa situação não é realista em todos os contextos de problemas de programação da produção, pois, existem muitas situações práticas em que os tempos das operações podem aumentar enquanto aguardam o processamento (CHRYSSOLOURIS, 2013).

Este fenômeno que ocorre com os problemas de escalonamento quando os tempos de processamento não são fixos é conhecido como o efeito de deterioração no tempo de processamento de tarefa. Isto ocorre quando a produtividade dos recursos (máquinas/trabalhadores) se deteriora ao longo do tempo, aumentando assim, os tempos reais de fabricação das tarefas. Assim, reduzir o desempenho dos recursos ao longo do tempo de processamento torna a representação do sistema mais realista e auxilia na construção de melhores modelos de tomada de decisão (MONTOYA-TORRES et al., 2021).

De acordo com a pesquisa realizada, Gupta e Gupta (1988) foram os primeiros autores que consideraram o conceito de deterioração em problemas de programação da produção. Eles mostraram o exemplo na indústria do aço, em que, se a temperatura de um lingote, esperando para entrar na máquina de rolamento, cair abaixo de certo nível, então, o lingote necessitará ser reaquecido antes de ser processado novamente. Eles consideram que os tempos de processamento das tarefas são uma função crescente de seus tempos de início (espera). O objetivo é minimizar o tempo total de conclusão do processamento das tarefas, definido como *makespan*, para um problema de uma única máquina.

Browne e Yechiali (1990) mostraram que as tarefas se deterioram, enquanto aguardam o processamento, fazendo com que o requisito de processamento aleatório de cada tarefa cresça a uma taxa específica da tarefa. Eles analisam os efeitos de diferentes

esquemas de deterioração e objetivam minimizar o *makespan* em uma única máquina.

Kunnathur e Gupta (1990) mostraram que o tempo e o esforço necessário para controlar um incêndio aumentam se há um atraso no início do combate ao incêndio. Eles consideram que o tempo de processamento de uma tarefa consiste em uma parte fixa e outra variável, sendo que a parte variável depende da hora de início da tarefa. O objetivo é minimizar o *makespan* para o problema em uma única máquina. Eles propuseram dois algoritmos, baseados em programação dinâmica e técnicas *branch-and-bound*.

Mosheiov (1991) considerou que as tarefas têm o mesmo tempo de processamento básico, mas o tempo de processamento real de cada tarefa cresce linearmente com seu tempo de início, e que uma taxa de crescimento diferente está associada a cada tarefa. O objetivo considerado foi minimizar o tempo de fluxo em uma única máquina. Em estudos semelhantes, Mosheiov (1994) objetivou minimizar o *makespan*, tempo de fluxo, atraso total, número de tarefas atrasadas em uma única máquina. Por sua vez, Mosheiov (1996) objetivou minimizar a soma dos tempos ponderados de conclusão, com pesos proporcionais aos tempos básicos de processamento em uma única máquina.

Outros exemplos podem ser encontrados na programação de manutenção, atribuições de limpeza, atribuições de trabalho da construção civil, unidades de emergência do hospital, alocação de recursos, processos químicos e metalúrgicos, em que um atraso no processamento de uma tarefa pode resultar em um esforço cada vez maior para realizá-la (MOSHEIOV, 2012).

Neste estudo, o problema tratado é uma variação dos problemas relacionados à programação da produção, conforme detalhado em Allahverdi (2015). Alinhado com as investigações conduzidas por Yang (2011) e Yang et al. (2012), o foco da pesquisa recai sobre cenários onde as máquinas estão sujeitas à deterioração, ou seja, não são as tarefas que sofrem deteriorações. Tal peculiaridade implica que os tempos de processamento das tarefas podem aumentar com base na sequência de operações nas máquinas (JOO; KIM, 2015). Esse acréscimo nos tempos é atribuído à deterioração progressiva das máquinas; em outras palavras, à medida que as máquinas executam as tarefas, seu desempenho tende a degradar (LEE et al., 2010).

Este capítulo tem por objetivo contextualizar o problema e se divide da seguinte maneira. A motivação para o estudo é apresentada na Seção 1.1. Em seguida, os objetivos alcançados na tese são apresentados na Seção 1.2. Na Seção 1.3 encontram-se as contribuições do trabalho. Por fim, a estrutura geral do texto é apresentada na Seção 1.4.

1.1 Motivação

É inegável o impressionante crescimento da oferta de materiais e produtos nos últimos anos, acompanhado pela expansão das vendas online que transformou a competição de local para global. Essa transformação trouxe consigo desafios cruciais para as indústrias, que agora devem enfrentar um mercado altamente competitivo. Como resultado, as empresas se viram compelidas a atender às crescentes expectativas dos consumidores, que incluem a busca por produtos de melhor qualidade, uma maior variedade de modelos, entregas confiáveis e rápidas, tudo isso a custos reduzidos. Gerenciar o processo de produção nesse cenário exige o cumprimento rigoroso das datas de entrega, a minimização dos tempos de processamento e de preparação, a redução do estoque e a otimização da utilização de recursos, sejam eles máquinas ou mão de obra (CHASE; JACOBS; AQUILANO, 2006).

Nesse cenário, os problemas de sequenciamento de tarefas (conhecidos como *scheduling*) desempenham um papel crucial, pois têm como objetivo principal a redução do tempo total de processamento em cada máquina, garantindo uma alocação eficiente dos recursos disponíveis. Uma das variantes desses problemas considera o fenômeno da deterioração que as tarefas podem causar nas máquinas. Essa variante tem recebido atenção crescente devido à sua relevância no contexto do gerenciamento logístico, uma vez que a deterioração das máquinas impacta diretamente no tempo final de processamento das tarefas.

De acordo com Kagermann et al. (2013), a Indústria 4.0 se caracteriza pela automação avançada, integração de sistemas e pelo extensivo uso de tecnologias digitais, representando uma transformação significativa na fabricação e nos processos industriais. Entende-se que o sequenciamento de tarefas em máquinas paralelas desempenha um papel crucial na Indústria 4.0. Em um ambiente de fabricação altamente automatizado e interconectado, o sequenciamento eficiente das tarefas em máquinas paralelas é essencial para otimizar a produção, minimizar tempos de espera e maximizar a utilização de recursos. A capacidade de tomar decisões de sequenciamento em tempo real, considerando fatores como a deterioração das máquinas e a programação de manutenção integrada, pode levar a uma produção mais ágil e econômica. Portanto, o aprimoramento das técnicas de sequenciamento de tarefas em máquinas paralelas é uma parte fundamental da transformação da Indústria 4.0, contribuindo para a eficiência operacional e a competitividade das empresas na era digital.

Este trabalho considera o problema de *scheduling* em máquinas paralelas não-relacionadas com efeito de deterioração. Este problema consiste em processar tarefas em máquinas paralelas levando em conta que cada tarefa executada provoca uma certa deterioração na máquina. Essa deterioração diminuirá o desempenho da máquina, aumentando o tempo de processamento da próxima tarefa a ser processada. Desta forma, as máquinas

se deterioram com base no conjunto de tarefas processado anteriormente e o tempo real para processar uma tarefa depende do nível de desempenho da máquina no início de cada tarefa. Ou seja, o tempo real de processamento de uma tarefa está em função das tarefas processadas anteriormente e dos seus efeitos de deterioração na máquina.

Ademais, esta pesquisa leva em consideração que a deterioração causada nas máquinas pode alcançar um nível crítico, resultando na falha delas. Quando uma máquina falha, sua capacidade de continuar a produção fica comprometida até que seja restaurada a um estado funcional por meio de uma operação de manutenção. O tempo de inatividade de uma máquina acarreta não apenas na perda de tempo de produção, mas também pode causar a paralisação de outras máquinas no sistema. Portanto, as operações de manutenção devem ser executadas com a maior brevidade possível para minimizar o tempo de inatividade da máquina, uma vez que o acúmulo de tempo de inatividade no sistema resulta em um impacto negativo crescente no desempenho (HOFFMAN et al., 2022). Nesse contexto, operações de manutenção são integradas ao sequenciamento com o objetivo de atenuar a deterioração das máquinas e, consequentemente, reduzir o tempo total de processamento.

Nos cenários apresentados, a relevância do problema abordado é evidente. Assim, podem-se destacar três motivos de grande importância que justificam a realização deste trabalho:

- Aplicabilidade prática do problema: o problema em questão possui aplicações práticas significativas.
- Complexidade do problema: o problema é notoriamente desafiador em termos de sua solução.
- Exploração limitada de abordagens multiobjetivo: a abordagem multiobjetivo para resolver o problema em questão ainda é pouco explorada na literatura.

Estes itens são detalhados a seguir:

Os problemas de sequenciamento que consideram deteriorações têm sido recentemente estudados devido às suas aplicações práticas no gerenciamento logístico nas indústrias, uma vez que a deterioração das máquinas afeta o tempo final da produção. O rendimento da produção de cada máquina pode diminuir devido a uma má posição das ferramentas, mal alinhamento das tarefas ou arranhadura de ferramentas. Nessas situações, o tempo de processamento das tarefas aumenta dependendo das posições em que as tarefas se encontram (JOO; KIM, 2015).

Um exemplo adicional do mundo real ocorre quando danos nas máquinas e a fadiga dos trabalhadores aumentam o tempo necessário para concluir uma tarefa. Setores como

construção e corte enfrentam esses desafios de forma frequente. No caso de ferramentas de corte, a degradação ao longo do tempo pode resultar em atrasos no subsequente processamento de materiais, muitas vezes exigindo a substituição das ferramentas. Da mesma forma, na indústria da construção, a fadiga dos trabalhadores pode ser aliviada por meio de pausas estratégicas, o que, por sua vez, reduz o tempo total necessário para concluir a sequência de tarefas (RUIZ-TORRES; PALETTA; M'HALLAH, 2017).

Outro exemplo do mundo real emerge quando os recursos, como os trabalhadores, perdem produtividade. Essa variação nos tempos de processamento é valiosa para modelar sistemas de produção com uma dependência significativa de mão-de-obra, como a indústria têxtil ou processos de separação. Nessas áreas, o tempo efetivo de processamento de uma tarefa tende a se deteriorar devido a uma série de fatores, como perda de produtividade, fadiga, exaustão, falta de motivação, estresse ou outros fenômenos semelhantes (SÁNCHEZ-HERRERA; MONTOYA-TORRES; SOLANO-CHARRIS, 2019).

O problema abordado está classificado como NP-difícil, como demonstrado por Ruiz-Torres, Paletta e Pérez (2013). Ele se reduz ao problema de sequenciamento de máquinas paralelas idênticas com o objetivo de minimizar o valor do *makespan*. É importante ressaltar que esse problema é amplamente reconhecido como NP-difícil, especialmente em instâncias em que as máquinas são idênticas e não há deterioração, como evidenciado por Bruno, Coffman e Sethi (1974), Garey e Johnson (1979), e Pinedo (2012). Portanto, para obter soluções de alta qualidade em um tempo computacional viável, é imperativo recorrer a algoritmos baseados em meta-heurísticas quando se lida com instâncias de médio e grande porte. As técnicas exatas, por outro lado, embora sejam geralmente impraticáveis em instâncias de médio e grande porte, podem ser aplicadas de forma eficiente no contexto de *matheuristics* (heurísticas matemáticas) para tratar instâncias maiores, especialmente em cenários reais (BOSCHETTI et al., 2009; SIMÕES; BATISTA; SOUZA, 2021).

Os estudos encontrados na literatura relacionados ao problema abordado (conforme detalhado na Seção 2.4) não adotam uma abordagem multiobjetivo, ou seja, não lidam com dois ou mais objetivos de forma simultânea. Assim, esta pesquisa busca preencher essa lacuna e estimular a realização de novas investigações nessa perspectiva.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é propor uma abordagem multiobjetivo e desenvolver um método eficaz para a resolução do problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas, considerando a deterioração da máquina dependente da sequência e a integração de operações de manutenção. Esse objetivo se alinha com a

busca por soluções que otimizem simultaneamente o *makespan* e o tempo total de atraso, enquanto também minimizam o tempo de inatividade das máquinas devido à manutenção.

1.2.2 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos foram estabelecidos:

(a) Investigar a literatura existente sobre problemas de sequenciamento com deterioração da máquina dependente da sequência e o escalonamento de operações de manutenção, a fim de compreender as abordagens existentes e as lacunas de pesquisa.

(b) Propor um modelo matemático para o problema de sequenciamento de tarefas biobjetivo que leve em consideração a deterioração da máquina dependente da sequência.

(c) Ajustar a formulação matemática proposta para integrar as operações de manutenção ao problema biobjetivo.

(d) Implementar algoritmos heurísticos de otimização multiobjetivo, incluindo um novo algoritmo híbrido, capaz de encontrar soluções eficientes para o problema proposto.

(e) Realizar experimentos computacionais abrangentes para avaliar o desempenho do algoritmo proposto em instâncias de diferentes tamanhos e complexidades, comparando-o com abordagens existentes na literatura.

(f) Analisar os resultados obtidos, a fim de fornecer *insights* para a otimização de sequenciamento de produção em ambientes com deterioração de máquina e operações de manutenção.

(g) Desenvolver técnicas para determinar as soluções ótimas do problema para instâncias de pequeno porte, a fim de validar os resultados obtidos.

Ao cumprir esses objetivos específicos, espera-se que este trabalho contribua para a solução de problemas de sequenciamento em sistemas de produção, proporcionando benefícios em termos de eficiência operacional, economia de recursos e melhor desempenho global.

1.3 Contribuições

Neste trabalho podem ser destacadas as seguintes contribuições:

(a) É proposto um novo modelo de programação não-linear inteiro misto, em inglês, *Mixed Integer Nonlinear Programming model* (MINLP) para o problema biobjetivo.

(b) É desenvolvido e validado um novo algoritmo híbrido que se baseia na abordagem de decomposição e agregação da meta-heurística *Multi-objective Evolutionary Algorithm based on Decomposition* (MOEA/D) e na busca local da meta-heurística *Iterated Local Search* (ILS), denominado *Iterated Local Search Based on Decomposition* (ILS/D).

(c) São apresentadas as soluções ótimas para as instâncias de pequeno porte.

(d) É proposto um novo MINLP para o problema biobjetivo com operações de manutenção integradas.

(e) É proposto um conjunto de instâncias para o problema multiobjetivo e operações de manutenção integradas em máquinas paralelas não-relacionadas.

(f) São propostas três estratégias para integrar as operações de manutenção no sequenciamento realizado pelos algoritmos, que permitem atingir soluções de programação de manutenção mais precisas para as instâncias de grande porte.

1.4 Estrutura do Trabalho

Este trabalho está dividido em capítulos que abordam os aspectos específicos do problema de sequenciamento de tarefas biobjetivo com deterioração da máquina dependente da sequência. A seguir, uma visão geral da organização do texto é apresentada para facilitar a compreensão da estrutura do trabalho como um todo.

- O Capítulo 1 desempenha um papel fundamental na contextualização do problema abordado neste trabalho. Este capítulo estabelece o cenário ao apresentar a motivação que impulsionou a pesquisa, delineia claramente os objetivos propostos e destaca as principais contribuições que este estudo oferece à área de estudo.
- O Capítulo 2 fornece uma revisão abrangente da literatura relacionada aos problemas de sequenciamento com efeito de deterioração, abordando três categorias distintas: deterioração dependente do tempo de início do processamento, dependente da posição e dependente da sequência de tarefas processadas.
- O Capítulo 3 apresenta a caracterização, o modelo matemático e um exemplo numérico tanto para o problema biobjetivo quanto para o problema integrado com as operações de manutenção.
- O Capítulo 4 discute os métodos de otimização multiobjetivo desenvolvidos neste trabalho, incluindo o algoritmo híbrido proposto para a resolução do problema. Além disso, são abordadas as estratégias para o sequenciamento das manutenções.

- O Capítulo 5 apresenta os experimentos computacionais realizados e análise detalhada dos resultados obtidos tanto para o problema biobjetivo quanto para o problema integrado com as operações de manutenção.
- Por fim, o Capítulo 6 encerra o trabalho, apresentando as conclusões alcançadas e delineando possíveis direções para futuras pesquisas na área.

Cada capítulo foi cuidadosamente elaborado para fornecer uma compreensão abrangente e detalhada do problema, das abordagens de resolução e dos resultados obtidos.

2 Revisão da Literatura

2.1 Introdução

Este capítulo apresenta os estudos que tratam dos problemas de *scheduling*, levando em consideração o impacto da deterioração. Na literatura, pelo menos três abordagens principais são identificadas para classificar esses problemas.

Na primeira abordagem, os modelos são sensíveis ao tempo de espera da tarefa antes do início de sua execução, denominada *time-dependent processing times* ou *time-dependent deterioration*, em inglês. Nesse cenário, o tempo de processamento de uma tarefa aumenta à medida que o seu início de execução é postergado, o que implica que quanto mais tardio for o início do processamento, maior será o tempo necessário para concluir a tarefa.

Na segunda abordagem, os tempos de processamento são dependentes da posição da tarefa no sistema de manufatura, conhecida como *position deterioration* ou *position-dependent*, em inglês. Nessa perspectiva, o tempo de processamento de uma tarefa está relacionado apenas ao número de tarefas já processadas ou à posição que uma tarefa ocupa no sequenciamento, como discutido por Mosheiov (2005) e Gawiejnowicz (2020).

Na terceira abordagem, a execução de uma tarefa provoca a deterioração da máquina, resultando em tempos de processamento mais longos para as tarefas subsequentes. Isso significa que os tempos de processamento dependem da sequência das tarefas previamente executadas, caracterizando a abordagem *sequence-dependent deterioration*, em inglês. Alguns trabalhos referentes às três abordagens são apresentados a seguir.

2.2 Deterioração Dependente do Tempo de Início do Processamento

No primeiro grupo, referido como *time-dependent deterioration*, o tempo de processamento atual de uma tarefa é definido por uma função do seu tempo de início. Esta função é dada por: $p'_{jk} = p_{jk} + d_{jk} \times \delta_{jk}$, onde p'_{jk}, p_{jk}, d_{jk} e δ_{jk} são o tempo de processamento atual, o tempo de processamento normal, a razão de deterioração e o tempo de início da tarefa j na máquina k, respectivamente. De acordo com esta abordagem, enquanto as tarefas esperam para serem processadas, o tempo de processamento atual das tarefas aumenta, isto é, o atraso no início do processamento de uma tarefa pode resultar em um tempo maior para processá-la (MAZDEH et al., 2010). Em outras palavras, estes trabalhos focam na deterioração da tarefa, em que as tarefas processadas em uma fase posterior requerem um tempo adicional em relação às tarefas processadas numa fase inicial.

A seguir são apresentados alguns dos trabalhos encontrados na literatura que seguem esta abordagem.

Toksarı e Güner (2009) abordam o problema de sequenciamento em máquinas paralelas considerando simultaneamente os efeitos de aprendizagem e de deterioração. O objetivo do problema é minimizar o adiantamento e o atraso para todas as tarefas que têm data de vencimento comum. Os autores desenvolveram um modelo matemático e um algoritmo para resolver os problemas de teste maiores. O algoritmo encontra boas soluções para problemas de 1.000 tarefas e 4 máquinas com 3 segundos em média. O desempenho do algoritmo é avaliado utilizando os resultados do modelo matemático.

Ji e Cheng (2009) analisam o problema de sequenciamento em máquinas paralelas em que o tempo de processamento de uma tarefa é uma função linear crescente de seu tempo de início. Os objetivos são minimizar o *makespan* e a carga total da máquina. Os autores mostram que o problema é NP-difícil. Para os testes com número arbitrário de máquinas, é provado que não existe nenhum algoritmo de aproximação de tempo polinomial capaz de resolvê-los. Quando o número de máquinas é fixo, são propostos dois algoritmos de aproximação de tempo polinomial.

Mazdeh et al. (2010) estudam o problema de programação em máquinas paralelas bi-critérios com efeitos de deterioração nas tarefas e máquinas. Os tempos de processamento das tarefas aumentam em funções de seus tempos de início e seguem uma deterioração linear simples. As funções objetivo são minimizar o atraso total e o custo de deterioração da máquina. Os autores desenvolvem um algoritmo para localizar as melhores soluções baseado na Busca Tabu.

Huang e Wang (2011) consideram o problema de sequenciamento de tarefas em máquinas paralelas idênticas com efeito de deterioração. Neste problema, os tempos de processamento das tarefas são definidos por funções de seus tempos de início. Os autores concentram em dois objetivos separadamente: minimizar o total de diferenças absolutas em tempos de conclusão e minimizar as diferenças totais absolutas em tempos de espera. Os autores mostram que os problemas são solucionáveis por algoritmos polinomiais.

Jiang (2011) aborda o problema de sequenciamento de tarefas em máquinas paralelas idênticas com produções em lote, no qual o tempo de processamento das tarefas é determinado por uma função de deterioração linear do tempo de início das tarefas. O objetivo é minimizar o maior tempo de processamento final das tarefas. Para resolver o problema, é proposto um algoritmo baseado na relaxação Lagrangiana.

Cheng et al. (2012) estudam o problema de sequenciamento em máquinas paralelas idênticas para minimizar o maior tempo de processamento final das tarefas, em que o tempo de processamento de cada tarefa é definido por uma função do tempo de início e uma data que é individual para todas as tarefas. Um modelo de programação inteira mista

é apresentado para o problema e são desenvolvidos um algoritmo modificado de busca em largura e um algoritmo *Variable Neighborhood Search* (VNS). Os resultados mostram que as abordagens propostas obtêm soluções próximas da ótima para instâncias de grande porte.

Huang, Wang e Ji (2014) consideram o problema de sequenciamento em máquinas paralelas idênticas com efeito de deterioração e o efeito de aprendizagem. O tempo de processamento da tarefa é dado por função do seu tempo de início e têm-se dois objetivos: minimizar uma função de custo que contém o maior tempo de processamento final e a diferença absoluta entre os tempos de processamentos finais das máquinas; e minimizar uma função de custo que contém o tempo de espera total e a diferença absoluta dos tempos de espera das tarefas para serem processadas nas máquinas.

Lalla-Ruiz e Voß (2016) abordam o problema de programação com deterioração das tarefas em etapas, buscando minimizar o tempo total de conclusão em máquinas paralelas idênticas onde cada tarefa possui uma data de deterioração e observa uma função para o tempo de processamento. Para resolvê-lo, foram propostos dois novos modelos matemáticos baseados no Problema de Particionamento de Conjuntos (*Set Partitioning Problem* - SPP) que melhoram o modelo proposto na literatura.

Gawiejnowicz (2020) realiza uma revisão abrangente das pesquisas realizadas nas últimas quatro décadas, que abordam o *time-dependent scheduling*, onde os tempos variáveis de processamento das tarefas dependem de quando as tarefas são iniciados. O trabalho está dividido em quatro partes: (i) mostra definições, noções, terminologias e os principais modelos; (ii) apresenta uma visão geral atualizada das quatro décadas de pesquisa de *time-dependent scheduling* com foco na complexidade computacional e nos algoritmos que resolvem esses problemas; (iii) concentra-se em novos tópicos, como problemas de dois agentes, mutuamente relacionados e jogos; (iv) discute os mais importantes problemas *time-dependent scheduling* que ainda aguardam solução.

Luo (2022) estuda problemas de escalonamento de máquina única em um ambiente estocástico onde os tempos de processamento e as datas de vencimento das tarefas são consideradas aleatórias. É proposto um novo modelo de deterioração dependente do tempo e um modelo de efeito de aprendizagem dependente da posição (função não-crescente de sua posição). Busca-se minimizar: os custos totais esperados do tempo de conclusão geral, os custos gerais totais esperados de atraso, os custos gerais máximos esperados do tempo de conclusão, o número total ponderado esperado de tarefas atrasadas e o esperado número total ponderado de tarefas atrasadas e adiantados. Vários cronogramas ótimos são obtidos.

2.3 Deterioração Dependente da Posição

O segundo grupo define que o tempo de processamento da tarefa está em função das posições das tarefas na sequência da máquina, isto é, o tempo de processamento atual da tarefa pode variar dependendo da posição em que uma tarefa está no sequenciamento, conhecido como *position-dependent*. Nesta abordagem, o tempo de processamento atual da tarefa j (se processada na posição h da máquina k) pode ser definido por: $p'_{jk} = p_{jk} + h \times d_{jk}$, onde d_{jk} é o efeito de deterioração da tarefa j na máquina k, e a posição h depende do número de tarefas depois de uma operação de manutenção (YANG, 2011). A operação de manutenção é frequentemente realizada no sistema de fabricação para diminuir o efeito de deterioração, com o objetivo de manter a eficiência da produção (LEE; YANG; YANG, 2013). Alguns trabalhos que seguem esta abordagem são apresentados em seguida.

Wang, Wang e Liu (2011) consideram o problema de sequenciamento de tarefas em máquinas paralelas idênticas com operação de manutenção deteriorante, isto é, se a manutenção das máquinas atrasar, mais tempo será necessário para realizá-la. O problema consiste em decidir a melhor sequência de tarefas e quando realizar a operação de manutenção para minimizar o maior tempo de processamento final das tarefas.

Da mesma forma, Wang e Wei (2011) tratam o problema de sequenciamento em máquinas paralelas idênticas com operações de manutenção. Nesta abordagem, os autores se concentram em dois objetivos: minimizar a diferença absoluta total do tempo final de processamento das tarefas e minimizar a diferença absoluta total do tempo de espera.

Yang (2011) aborda o problema de sequenciamento em máquinas paralelas idênticas com operações de manutenção e efeitos de deterioração dependentes da posição. Assume-se que a operação de manutenção deve ser realizada exatamente uma vez em cada máquina e logo após o término do processamento de qualquer tarefa. Além disso, após a manutenção, a máquina reverte para sua condição inicial e os efeitos de deterioração iniciam-se. O objetivo do problema é encontrar a melhor sequência das tarefas e a melhor posição das operações de manutenção para minimizar a carga total da máquina.

Yang et al. (2012) abordam o problema de sequenciamento em máquinas paralelas não-relacionadas considerando simultaneamente operações de manutenção e efeitos de aprendizagem (entende-se como o oposto da deterioração, ou seja, o processamento de uma tarefa pode encurtar a duração da tarefa subsequente). O objetivo é encontrar a frequência de manutenção, as posições ótimas para as operações de manutenção e a sequência ótima de tarefas para minimizar a carga total da máquina. Os autores aplicaram balanceamento para obter as posições ótimas das operações de manutenção e o número de tarefas em cada grupo do sequenciamento de cada máquina. Além disso, um algoritmo foi desenvolvido para resolver o problema quando a frequência de manutenção das máquinas é dada.

Hsu et al. (2013) consideram o problema de sequenciamento em máquinas paralelas

não-relacionadas com efeitos de aprendizagem e operações de manutenção simultaneamente. A duração da operação de manutenção é uma função linear do seu tempo de início. O objetivo do problema é minimizar o maior tempo de processamento final da máquina. Os autores desenvolveram três tipos de modelos para o efeito de aprendizagem e mostram que os três modelos propostos podem ser resolvidos de forma ótima em tempo polinomial.

Hsu e Yang (2014) tratam do problema de sequenciamento de alocação de recursos, em máquinas paralelas não-relacionadas, com efeito de deterioração dependente da posição. Os objetivos são minimizar a carga total, o maior tempo de processamento final, o desvio absoluto total dos tempos de processamentos finais e o custo total do recurso. Além disso, esta abordagem visa à redução da carga total, do tempo de espera total, do desvio absoluto total do tempo de espera e do custo total do recurso. Os autores mostram que o problema é resolvido em tempo polinomial quando o número de máquinas é fixado.

Gara-Ali, Espinouse e Finke (2014) introduzem um modelo geral para o problema de sequenciamento em máquinas paralelas não-relacionadas com o efeito de deterioração dependente da posição e de múltiplas operações de manutenção em cada máquina. O objetivo é encontrar juntamente a posição ótima da operação de manutenção, as frequências das manutenções e minimizar os critérios de desempenho. Os autores desenvolvem um algoritmo em tempo polinomial para o problema quando o número de máquinas é fixado.

Ma et al. (2015) consideram o sequenciamento em máquinas paralelas com tempos de entrega das tarefas dependentes da sequência já processada e da operação de manutenção. Os autores consideram três versões do problema para minimizar a diferença total absoluta dos tempos de conclusão das tarefas, a carga total em todas as máquinas e o tempo total de conclusão.

Sánchez-Herrera, Montoya-Torres e Solano-Charris (2019) abordam um problema de sequenciamento de *flow shop* com tempos de processamento dependentes de posição. O objetivo é minimizar o atraso máximo e analisar os efeitos do fator de deterioração. São propostos um modelo matemático, heurísticas e meta-heurísticas. Os resultados mostram a eficiência das meta-heurística s e fornecem *insights* sobre o impacto do fator de deterioração em problemas de sequenciamento de *flow shop*.

Pei et al. (2021) abordam um problema de *serial-batching* em máquinas paralelas com o objetivo de minimizar o *makespan*. Os efeitos da aprendizagem e da deterioração são considerados simultaneamente, e o tempo real de processamento de cada tarefa depende da soma dos tempos de processamento das tarefas anteriores e da posição da tarefa atual. Foi proposto um novo algoritmo híbrido SC-VNS, que combina o algoritmo *Society and Civilization* (SC) com *Variable Neighborhood Search* (VNS). Os resultados dos experimentos demonstraram que o algoritmo SC-VNS tem um desempenho bastante melhor do que os algoritmos comparados em termos de qualidade da solução e tempo de execução necessário.

Liu et al. (2022) lidam com um problema de sequenciamento de máquina única com uma operação de manutenção opcional, onde o tempo real de processamento de uma tarefa está em função de sua posição e seu tempo de início. O objetivo é determinar uma sequência de tarefa ideal e a localização da operação de manutenção de forma que o *makepan* seja minimizado. Para resolver o problema, desenvolveram um algoritmo de tempo polinomial, em que a complexidade de tempo é $O(n^4)$, onde n é o número de tarefas.

2.4 Deterioração Dependente da Sequência de Tarefas Processadas

Neste grupo de trabalhos, encontra-se o problema de sequenciamento de deterioração da máquina dependente da sequência, conhecido como *sequence-dependent deterioration*, em que o tempo real de processamento da tarefa depende do conjunto de tarefas previamente processado pela máquina e seu efeito de deterioração na máquina. De acordo com a pesquisa realizada, Ruiz-Torres, Paletta e Pérez (2013) consideraram pela primeira vez o conceito de deterioração dependente de sequência em problemas de sequenciamento, desde então, vários autores têm estudado este problema. Os trabalhos encontrados na literatura foram organizados em uma tabela para melhor visualização dos objetivos e métodos abordados por cada autor. A Tabela 1 apresenta estes trabalhos que foram publicados na literatura entre 2013 e 2022.

Tabela 1 – Revisão de literatura para problemas de *scheduling* com deterioração dependente de sequência.

Referência	Objetivo	Método
Ruiz-Torres, Paletta e Pérez (2013)	Minimizar o *makespan* (mono-objetivo).	Eles mostraram que para uma única máquina o problema pode ser resolvido em tempo polinomial, para duas ou mais máquinas o problema é NP-difícil. Eles propuseram um modelo *Mixed Integer Nonlinear Programming* (MINLP) e uma meta-heurística *Simulated Annealing* (SA*) para instâncias de tamanho pequeno ($n \in \{8, 11, 14\}$ e $m \in \{2, 7, 4\}$) e instâncias de tamanho médio ($n \in \{20, 35, 50\}$ e $m \in \{4, 7, 10\}$).
Ruiz-Torres, Paletta e Perez-Roman (2015)	Maximizar a porcentagem de tarefas *on-time*, ou seja, minimizar o número de tarefas atrasadas (mono-objetivo).	Eles propuseram um modelo MINLP e um conjunto de instâncias do problema que considera vários níveis de número de tarefas e máquinas, a data de vencimento e o efeito de deterioração. Duas estratégias são propostas: uma carrega uma máquina por vez e outra considera todas as máquinas simultaneamente. As duas estratégias são combinadas com um conjunto de algoritmos de sequenciamento para resolver o problema de máquinas idênticas e não-relacionadas.

Santos, Arroyo e Carvalho (2016)	Minimizar o *makespan* (mono-objetivo).	Eles propuseram uma meta-heurística ILS combinada com uma busca local *Random Variable Neighborhood Descent* (RVND), denominada ILS-RVND. Eles mostraram que ILS-RVND superou computacionalmente a heurística SA* proposta em Ruiz-Torres, Paletta e Pérez (2013) em instâncias de tamanho médio ($n \in \{20, 35, 50\}$ e $m \in \{4, 7, 10\}$) e instâncias de grande porte ($n \in \{80, 100, 150\}$ e $m \in \{5, 10, 20\}$), propostas no trabalho.
Santos e Arroyo (2017)	Minimizar o *makespan* (mono-objetivo).	Eles propuseram uma heurística baseada na metaheurística *Iterated Greedy* (IG) combinada ao método RVND, denominada IG-RVND. IG-RVND é comparado com algoritmos SA*, ILS e IG básico. Os resultados mostram que o IG-RVND supera os algoritmos concorrentes nas instâncias de médio e grande porte.
Araújo, Dhein e Fampa (2017)	Minimizar o *makespan* (mono-objetivo).	Eles aplicaram uma estratégia de linearização para construir uma formulação de programação linear inteira mista para o problema proposto em Ruiz-Torres, Paletta e Pérez (2013). Eles resolveram as instâncias de tamanho médio ($n \in \{20, 35, 50\}$ e $m \in \{4, 7, 10\}$) de forma ótima, ou seja, exata, pela primeira vez.
Ruiz-Torres, Paletta e M'Hallah (2017)	Minimizar o *makespan* (mono-objetivo).	Eles introduziram operações de manutenção preventiva de duração determinística conhecida, para reduzir o efeito de deterioração na máquina e consequentemente o tempo total de conclusão das tarefas atribuídas às máquinas. Eles propuseram um modelo MINLP e apresentaram três heurísticas construtivas, que foram testadas em 600 instâncias de máquinas paralelas idênticas, para encontrar soluções de boa qualidade para o problema NP-difícil.
Pérez, Ambati e Ruiz-Torres (2018)	Maximizar a porcentagem de tarefas *on-time*, ou seja, minimizar o número de tarefas atrasadas (mono-objetivo).	Eles estudaram o problema em que operações de manutenção periódicas são executadas em máquinas paralelas idênticas e não-relacionadas. Eles propuseram um modelo *Mixed Integer Programming* (MIP) e duas heurísticas para resolver o problema: a primeira é baseada no carregamento de uma máquina por vez, e a segunda considera todas as máquinas simultaneamente.

Ding et al. (2019)	Minimizar o *makespan* / minimizar tempo de conclusão ponderado (mono-objetivo).	Eles apresentaram um algoritmo *Ejection Chain* para minimizar dois objetivos independentes, ou seja, um objetivo de cada vez: *makespan* e o tempo total de conclusão ponderado. Eles alcançaram todas as soluções ótimas com uma taxa de acerto de 100% para instâncias de tamanho pequeno e melhoraram os resultados conhecidos para 388 instâncias de tamanho médio.
Delorme, Iori e Mendes (2021)	Minimizar o *makespan* (mono-objetivo).	Eles estudaram o mesmo problema com operações de manutenção, proposto em Ruiz-Torres, Paletta e M'Hallah (2017). Eles introduziram quatro modelos de programação linear inteira mista, em dois modelos usaram restrições big-M e os outros dois usaram um número exponencial de variáveis. Foi desenvolvida uma heurística ILS para lidar com um conjunto de 480 instâncias de grande porte envolvendo máquinas paralelas não-relacionadas. O desempenho das abordagens propostas foi avaliado por meio de experimentos computacionais abrangentes.
Ruiz-Torres, Ablanedo-Rosas e Melnik (2021)	Minimizar o *makespan* / minimizar o atraso médio (mono-objetivo).	Eles abordam o problema *Flow-shop* de permutação em duas máquinas para minimizar o *makespan* e o atraso médio, em uma abordagem mono-objetivo, ou seja, um objetivo de cada vez. Eles propuseram múltiplas heurísticas e os resultados indicam que, em conjunto, as heurísticas geram soluções muito próximas da ótima para ambos os critérios.
Santos et al. (2022a)	Minimizar o *makespan* e minimizar o atraso total na entrega das tarefas (multiobjetivo).	Eles apresentaram uma abordagem inovadora para o problema, introduzindo pela primeira vez uma abordagem multiobjetivo. Propuseram um modelo MINLP que visa a minimização simultânea dos dois objetivos e desenvolveram uma heurística híbrida denominada ILS/D. O algoritmo ILS/D combina os princípios de decomposição e agregação do MOEA/D com o mecanismo de busca local do ILS. Realizaram uma comparação entre os resultados obtidos pelos algoritmos ILS/D, MOEA/D e *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) e os resultados indicaram que o desempenho do ILS/D superou significativamente o do MOEA/D e do NSGA-II.

Santos et al. (2022b)	Minimizar o *makespan* e minimizar o atraso total na entrega das tarefas (multiobjetivo).	Eles compararam duas abordagens distintas para estender a meta-heurística ILS ao problema multiobjetivo: o *Pareto Iterated Local Search* (PILS), uma extensão direta da heurística *Pareto Local Search* (PLS), que emprega busca local multiobjetivo, e o algoritmo híbrido ILS/D. A avaliação do desempenho do PILS, que se baseia exclusivamente em meta-heurística de busca local, foi realizada, considerando que o problema já havia sido abordado em algoritmos populacionais e meta-heurísticas híbridas. Experimentos numéricos foram conduzidos em problemas de teste de diversas escalas. Os resultados experimentais apontam que a heurística ILS/D geralmente supera o PILS em uma variedade de cenários.

2.5 Conclusão

Como evidenciado, o problema multiobjetivo foi previamente abordado por Santos et al. (2022a) e Santos et al. (2022b), resultando em contribuições significativas que emanaram deste estudo. No entanto, a problemática associada às operações de manutenção recebeu atenção em trabalhos anteriores, como os de Ruiz-Torres, Paletta e M'Hallah (2017), Pérez, Ambati e Ruiz-Torres (2018), e Delorme, Iori e Mendes (2021), que se concentraram predominantemente em formulações de objetivo único. Assim, este estudo visa preencher uma lacuna de pesquisa crucial, ao abordar o problema multiobjetivo em conjunto com o escalonamento das operações de manutenção.

Dessa forma, é importante estudar o problema de sequenciamento com deterioração da máquina dependente da sequência integrado às operações de manutenção. Isso ocorre porque os sequenciamentos desempenham um papel central nos sistemas de fabricação modernos, desafiando-nos a encontrar soluções que minimizem o tempo necessário para concluir todos os trabalhos de forma eficiente e eficaz (ZHENG et al., 2021).

3 Problema de Sequenciamento de Tarefas Biobjetivo com a Deterioração da Máquina Dependente da Sequência

3.1 Introdução

Este capítulo aborda a complexidade do problema de sequenciamento em cenários envolvendo máquinas paralelas não-relacionadas, tempos de processamento dependentes da sequência de tarefas processadas, deterioração das máquinas também dependente dessa sequência, e a integração das operações de manutenção.

Nesse contexto, é importante destacar que os tempos efetivos de processamento de uma tarefa podem variar devido à deterioração gradual da máquina, sendo influenciados pelas tarefas processadas anteriormente. O ato de processar uma tarefa resulta em um leve declínio no desempenho da máquina para as tarefas subsequentes. Além disso, é considerado que a execução de uma operação de manutenção, denominada "tarefa de manutenção", tem a capacidade de restaurar completamente o desempenho da máquina afetada. Essa restauração visa reduzir a deterioração das máquinas e, consequentemente, mitigar o tempo total de processamento.

O objetivo principal deste capítulo é propor uma abordagem multiobjetivo para o problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas considerando a deterioração da máquina dependente da sequência.

3.2 Sequenciamento com a Deterioração da Máquina Dependente da Sequência

Esta seção aborda o problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas, em que as máquinas sofrem deterioração baseado no conjunto de tarefas processado anteriormente. O problema tem dois objetivos, que neste trabalho são considerados simultaneamente. O primeiro é encontrar a sequência de processamento das tarefas em cada máquina para minimizar o tempo máximo para conclusão de todas as tarefas, conhecido como *makespan*. E o segundo objetivo é minimizar o tempo total de atraso na entrega das tarefas.

As próximas seções apresentam as características do problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas considerando a deterioração da máquina dependente da sequência, com uma abordagem biobjetivo.

3.2.1 Caracterização do Problema

O presente problema é definido por Ruiz-Torres, Paletta e Pérez (2013), e Ruiz-Torres, Paletta e Perez-Roman (2015), da seguinte forma:

(a) Existe um conjunto de n tarefas independentes, $N = \{1, \ldots, j, \ldots, n\}$, para serem processadas em um conjunto de m máquinas paralelas, $M = \{1, \ldots, k, \ldots, m\}$.

(b) Todas as tarefas estão disponíveis para o processamento no tempo zero, são não-preemptivas, isto é, a execução de cada tarefa não pode ser interrompida até que seja totalmente concluída e cada tarefa deve ser processada uma única vez.

(c) Cada máquina pode processar somente uma tarefa de cada vez, não pode ficar ociosa até a última tarefa atribuída ter sido finalizada e estas máquinas são classificadas como máquinas paralelas não-relacionadas, isto é, cada tarefa j possui um tempo de processamento que depende da máquina k na qual será alocada.

(d) O tempo de processamento normal da tarefa j na máquina k é definido por p_{jk}.

(e) O efeito de deterioração da tarefa j na máquina k é dado por d_{jk}, em que, $0 \leq d_{jk} \leq 1$ $\forall j \in N$ e $k \in M$.

(f) O desempenho da máquina k na posição h (isto é, após executar a tarefa que está na posição $h-1$) é determinado por:

$$q_{k[h]} = (1 - d_{[h-1]k}) \times q_{k[h-1]} \tag{3.1}$$

em que $d_{[h-1]k}$ é o efeito de deterioração provocado na máquina k pela execução da tarefa que está na posição $h-1$.

(g) Assume-se que todas as máquinas iniciam sem efeito de deterioração, sendo assim, o nível de desempenho para processar a tarefa na posição 1 é 100%, ou seja, $q_{k[1]} = 1, \forall k \in M$.

(h) O tempo de processamento atual de uma tarefa j processada na máquina k na posição h é determinado por:

$$p'_{jk} = \frac{p_{jk}}{q_{k[h]}} \tag{3.2}$$

(i) Existem n possíveis posições em cada máquina, e A é definido como o conjunto de posições.

(j) Cada tarefa j possui uma data de entrega e_j.

O objetivo do trabalho de Ruiz-Torres, Paletta e Pérez (2013) é minimizar o *makespan*. E o objetivo do trabalho de Ruiz-Torres, Paletta e Perez-Roman (2015) é maximizar a porcentagem de tarefas entregues dentro do prazo. O principal diferencial deste trabalho é a presença de dois objetivos a serem otimizados simultaneamente. O primeiro objetivo do problema é minimizar o tempo máximo de conclusão das tarefas, também conhecido como *makespan* (C_{max}). O segundo objetivo do problema é minimizar o tempo total de atraso na entrega das tarefas, em inglês, *total delay time* (TDT). De acordo com a classificação de Graham et al. (1979), este problema pode ser denotado por $Rm|Sdd|(C_{max}, TDT)$, onde Rm significa o ambiente de máquinas não-relacionadas, Sdd corresponde às restrições de deterioração dependente da sequência, C_{max} e TDT correspondem aos critérios de otimização, em que C_{max} refere-se ao *makespan* e TDT ao tempo total de atraso na entrega das tarefas.

3.2.2 Modelo Matemático

Nesta seção, é apresentado o modelo de programação não-linear inteiro misto biobjetivo do problema $Rm|Sdd|(C_{max}, TDT)$, baseado nos problemas mono-objetivo definidos por Ruiz-Torres, Paletta e Pérez (2013), e Ruiz-Torres, Paletta e Perez-Roman (2015).

No modelo matemático para o problema $Rm|Sdd|(C_{max}, TDT)$, usam-se as seguintes variáveis de decisão:

- C_{max}: máximo tempo de processamento de todas as máquinas (*makespan*).
- t_{kh}: tempo de atraso da tarefa atribuída à máquina k na posição h
- q_{kh}: desempenho da máquina k na posição h.
- $x_{jkh} = \begin{cases} 1, \text{ se a tarefa } j \text{ é atribuída para a máquina } k \text{ na posição } h \\ 0, \text{ caso contrário.} \end{cases}$

O modelo matemático biobjetivo do problema é dado por:

$$\min C_{max} \tag{3.3}$$

$$\min TDT = \sum_{h \in A} \sum_{k \in M} \sum_{j \in N} t_{kh} \times x_{jkh} \tag{3.4}$$

$$\sum_{j \in N} x_{jkh} \leq 1, \forall h \in A, k \in M \tag{3.5}$$

$$\sum_{h \in A} \sum_{k \in M} x_{jkh} = 1, \forall j \in N \tag{3.6}$$

$$\sum_{j \in N} \sum_{h \in A} \frac{p_{jk}}{q_{kh}} \times x_{jkh} \leq C_{max}, \forall k \in M \tag{3.7}$$

$$x_{jkh} \leq \sum_{l \in N} x_{lk(h-1)}, \forall j \in N, k \in M, h \in A \backslash \{1\} \tag{3.8}$$

$$q_{kh} = \sum_{j \in N} (1 - d_{jk}) \times q_{k(h-1)} \times x_{jk(h-1)}, \forall h \in A \backslash \{1\}, k \in M \tag{3.9}$$

$$q_{k1} = 1, \forall k \in M \tag{3.10}$$

$$t_{kh} \geq \sum_{j \in N} \sum_{l=1...h} \frac{p_{jk}}{q_{kl}} \times x_{jkl} - \sum_{j \in N} e_j \times x_{jkh}, \forall k \in M, h \in A \tag{3.11}$$

$$t_{kh} \geq 0, \forall k \in M, h \in A \tag{3.12}$$

$$x_{jkh} \in \{0, 1\}, \forall j \in N, k \in M, h \in A \tag{3.13}$$

No modelo, o primeiro objetivo é minimizar o C_{max} (*makespan*) definido na equação (3.3) e o segundo objetivo é minimizar o tempo total de atraso nas entregas das tarefas (TDT) definido na equação (3.4). As restrições (3.5) garantem que cada posição em cada máquina pode ter no máximo uma tarefa atribuída e as restrições (3.6) garantem que cada tarefa deve ser atribuída para exatamente uma posição em uma máquina. As restrições (3.7) garantem que a carga total em cada máquina seja igual ou menor que C_{max}. As restrições (3.8) garantem atribuições contínuas de tarefas. Por exemplo, dada uma máquina que possui tarefas nas posições $h = 1$, $h = 2$ e $h = 3$, esta restrição impede que um nova tarefa seja inserida na posição $h \geq 5$, pois não há tarefa na posição $h = 4$, mantendo assim uma atribuição contínua na máquina. As equações (3.9) e (3.10) definem o nível de desempenho em cada posição da máquina. Para a primeira posição, todas as máquinas têm desempenho igual a um, ou seja, 100% de produtividade. A equação (3.11) define o tempo de atraso da tarefa atribuída à máquina k na posição h. Se o valor de $t_{kh} = 0$, isso indica que a tarefa está dentro do prazo, enquanto $t_{kh} > 0$ indica que a tarefa naquela posição/máquina está atrasada. A equação (3.12) garante que o tempo de atraso das tarefas seja maior ou igual a zero. Finalmente, a Equação (3.13) define o valor binário para a variável x_{jkh}.

Pode-se perceber que este modelo de programação inteira é não-linear por causa da expressão $\frac{x_{jkh}}{q_{kh}}$ nas restrições (3.7) e (3.11); por causa da expressão $q_{kh} \times x_{jkh}$ nas restrições (3.9) e por causa da expressão $t_{kh} \times x_{jkh}$ na segunda função objetivo (3.4). Conforme demonstrado na Seção 2.4, Araújo, Dhein e Fampa (2017) apresentaram uma formulação de programação linear inteira mista para um problema mono-objetivo semelhante. É importante observar que, nesse contexto, a linearização foi aplicada apenas às restrições do problema. Entretanto, no cenário atual, a função objetivo é não-linear, o que impede a obtenção de soluções ótimas para o problema em questão, mesmo se fosse utilizado a estratégia de linearização proposta por Araújo, Dhein e Fampa (2017). Vale ressaltar também que foram feitas tentativas de resolver o modelo usando um solver para problemas

Tabela 2 – Uma instância do problema de sequenciamento com deterioração da máquina dependente de sequência, com $n = 10$ e $m = 2$.

Tarefa	p_{j1}	p_{j2}	d_{j1}	d_{j2}	e_j
1	59,0	76,0	0,0401	0,0341	157,0
2	10,0	08,0	0,0306	0,0353	26,0
3	92,0	90,0	0,0348	0,0365	185,0
4	30,0	22,0	0,0298	0,0497	78,0
5	71,0	45,0	0,0415	0,0469	194,0
6	82,0	93,0	0,0368	0,0489	140,0
7	63,0	50,0	0,0146	0,0265	119,0
8	75,0	54,0	0,0189	0,0425	154,0
9	42,0	23,0	0,0277	0,0347	199,0
10	46,0	37,0	0,0287	0,0220	145,0

não-lineares, o SCIP[1] (*Solving Constraint Integer Programs*)(GAMRATH et al., 2020), por meio da biblioteca PySCIPOpt[2]. No entanto, devido à complexidade do modelo, o solver testado não conseguiu resolver todas as instâncias de pequeno porte em um tempo computacional viável. Tornando-se necessário recorrer a uma técnica de enumeração, como é apresentado na Seção 5.2.6.

3.2.3 Exemplo Numérico

Para mostrar as características do problema em estudo, a seguir é apresentado um exemplo para uma instância do problema com $n = 10$ tarefas e $m = 2$ máquinas. A Tabela 2 apresenta os valores de tempos de processamento (p_{jk}), efeito de deterioração (d_{jk}) de cada tarefa j em cada máquina k, e na última coluna, as datas de entregas de cada tarefa j são apresentadas.

A Figura 1 mostra dois possíveis sequenciamentos, nos quais é apresentada a sequência de tarefas que é atribuída para cada máquina. Para cada posição, é exibido: o nível de desempenho da máquina ao iniciar a execução da tarefa (acima), o tempo de processamento atual ($p'_{j,k}$), o tempo de atraso da tarefa (t_{kh}) e o tempo total de processamento, no final de cada tarefa (abaixo).

No primeiro sequenciamento, as tarefas 6, 1, 3 e 4 são atribuídas à máquina 1 e as tarefas 2, 7, 10, 9, 5 e 8 são atribuídas à máquina 2. Para encontrar o tempo de processamento atual das tarefas é necessário calcular o desempenho da máquina após a execução de uma tarefa. Observa-se que as máquinas m_1 e m_2, na primeira posição, executam as tarefas 6 e 2, respectivamente, com 100% de desempenho. Estas primeiras

[1] https://www.scipopt.org/
[2] https://pypi.org/project/PySCIPOpt/

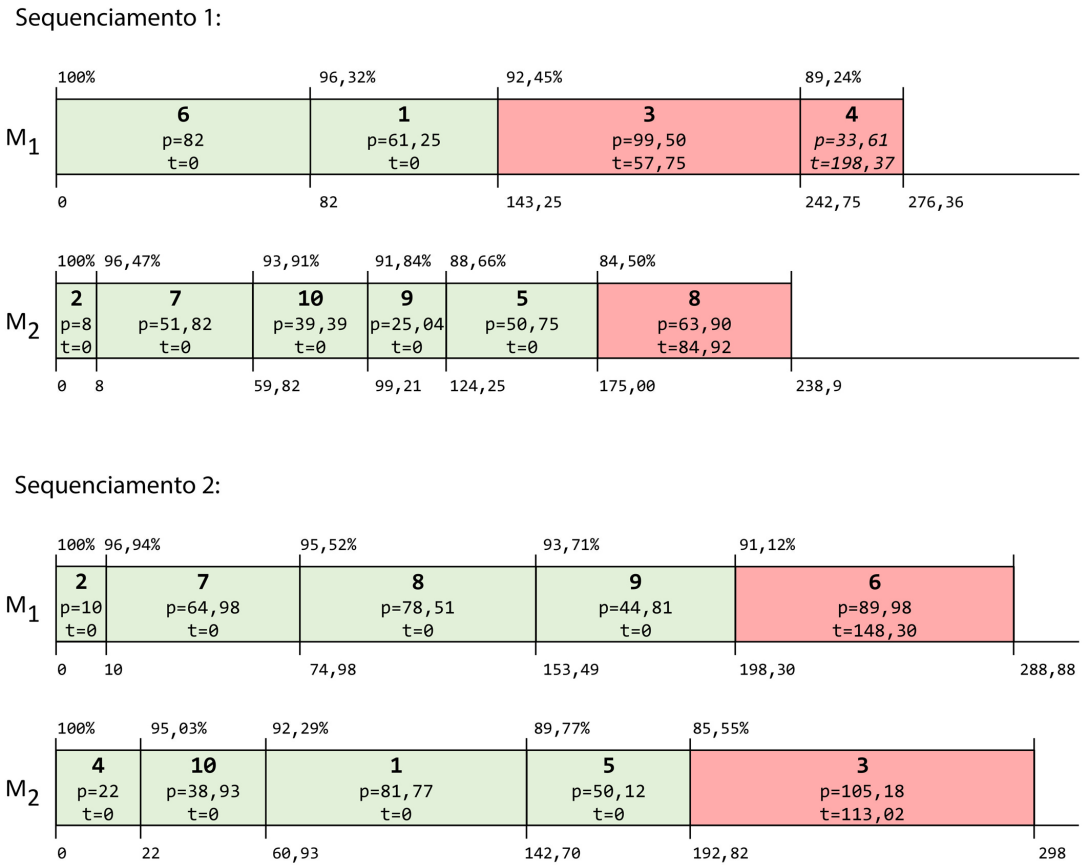


Figura 1 – Sequenciamentos para 10 tarefas e 2 máquinas.

tarefas causam deteriorações nas máquinas, diminuindo o seu desempenho. Por exemplo, a tarefa 6 deteriorou a máquina 1, então o novo desempenho desta máquina na posição $h = 2$ (após executar a tarefa 6) é obtido pela Equação (3.1) da seguinte maneira: $q_{1[2]} = (1 - d_{[1]1}) \times q_{1[1]} = (1 - 0,0368) \times 1 = 0,9632 = 96,32\%$. Como o desempenho da máquina diminuiu, então, a tarefa que ocupa a segunda posição será executada em um tempo maior. O tempo de processamento atual para a tarefa 1, que ocupa a posição $h = 2$ na máquina 1, é obtido da seguinte maneira: $\frac{59,0}{0,9632} = 61,25$.

A redução do desempenho das máquinas pode ser facilmente notada comparando a soma dos tempos de processamento para as quatro tarefas da máquina 1 sem efeito de deterioração (263 unidades de tempo). Porém, no sequenciamento, o tempo total necessário para concluir as tarefas é igual a 276,36 unidades de tempo, dado o efeito de deterioração (exemplo levando em conta o sequenciamento 1, da Figura 1).

Os dois sequenciamentos têm valores diferentes para as funções objetivo em consideração. No sequenciamento 1, o resultado do TDT é igual a 341,04 unidades de tempo, enquanto no sequenciamento 2, o resultado do TDT é menor, igual a 261,32 unidades de tempo.

O *makespan* (276,36) é menor no sequenciamento 1 (que possui TDT maior). Por outro lado, o sequenciamento 2 tem o *makespan* (298) maior e o TDT menor. Portanto,

existe um *tradeoff* quando se considera o *makespan* versus o tempo total de atraso das tarefas entregues. Além disso, os sequenciamentos fornecem valores diferentes para outros critérios de relevância. Por exemplo, tanto o *makespan* quanto a soma dos tempos de conclusão das máquinas são menores no sequenciamento 1 (que tem um tempo total de atraso maior). E tanto o tempo de atraso total quanto o número de tarefas atrasadas são menores no sequenciamento 2 (que tem o valor de *makespan* e a soma dos tempos de conclusão das máquinas maiores).

3.3 Sequenciamento com a Deterioração da Máquina Dependente da Sequência e Manutenção Integrada

Esta seção aborda o problema de sequenciamento envolvendo a deterioração de máquina dependente de sequência e operações de manutenção integradas. Uma operação de manutenção (tratada, neste trabalho, como "tarefa de manutenção") pode reparar a deterioração das máquinas, ou seja, o desempenho das máquinas pode ser totalmente restaurado após realizar a manutenção. Este problema é comumente visto no mundo real, onde o tempo de processamento das tarefas pode ser estendido devido aos danos causados nas máquinas e à fadiga dos trabalhadores. As indústrias de construção e corte são exemplos deste tipo de atividade (RUIZ-TORRES; PALETTA; M'HALLAH, 2017).

Devido à dureza do material que está sendo cortado, as ferramentas de corte podem se deteriorar com o tempo. Assim, pode haver atraso no corte dos materiais subsequentes no agendamento e as ferramentas de corte podem ser severamente danificadas, necessitando de substituição. Portanto, realizar manutenção nessas ferramentas pode reduzir o tempo necessário para concluir a sequência de tarefas.

Na indústria da construção, uma situação semelhante pode ser observada. A dificuldade de um trabalho pode afetar o nível de fadiga dos trabalhadores da construção civil, resultando em jornadas de trabalho mais longas e na necessidade de troca de trabalhadores em alguns casos. Ao permitir que os trabalhadores tirem folga, você pode reduzir a fadiga do trabalhador e garantir que o sequenciamento de tarefas seja concluído em um período de tempo mais curto.

Os problemas que envolvem programação de manutenções têm sido amplamente estudados na literatura. Dentre esses, destacam-se os problemas de *scheduling* que levam em consideração o impacto da deterioração dependente da posição das tarefas e a programação das manutenções, abordados nas pesquisas conduzidas por Yang (2011) e Yang et al. (2012). Além disso, outro ramo de estudos com contribuições importantes na literatura, abrange os problemas de *scheduling* que incorporam manutenções preventivas. Neste contexto, podem ser destacados os trabalhos de Aquino, Chagas e Souza (2019), Levitin, Xing e Dai (2021), e Alfares (2022). Essas pesquisas abordam diferentes aspectos da programação de

manutenções e seus impactos nos sistemas de produção.

Esta seção tem como objetivo aprimorar a pesquisa realizada por Ruiz-Torres, Paletta e M'Hallah (2017) e Santos et al. (2022a), integrando as operações de manutenção ao problema multiobjetivo. De acordo com o estudo realizado, este problema de pesquisa ainda não foi abordado na literatura. Dessa forma, entende-se que a pesquisa que considera minimizar, simultaneamente, o *makespan* e o atraso total das datas de entrega das tarefas, e conjuntamente aborda como planejar manutenções para reduzir o efeito da deterioração da máquina dependente de sequência é inexistente na literatura. Preencher essa lacuna é importante uma vez que auxilia na tomada decisões dos sistemas de produção e facilita o desenvolvimento de soluções que podem impactar nos resultados e maximizar a produtividade e eficiência do processo de produção.

Considerando um ambiente em que os equipamentos deterioram-se, uma operação de manutenção pode reduzir o tempo total de conclusão das tarefas atribuídos à máquina, uma vez que recupera o desempenho do sistema. Ou seja, uma operação de manutenção restaura a deterioração da máquina causada pelo conjunto de tarefas processadas desde a última manutenção realizada. As tarefas de manutenção podem ser escalonadas várias vezes em uma mesma máquina, e podem ser realizadas simultaneamente em qualquer máquina. Contudo, por se tratar de máquinas paralelas não-relacionadas, isto é, não idênticas, cada máquina possui uma duração do tempo de manutenção.

3.3.1 Caracterização do Problema

O presente problema é semelhante ao definido na seção anterior (Seção 3.2), porém são adicionados os itens das letras (k) até (o) relacionados às operações de manutenção. Estas novas definições foram baseadas no trabalho de Ruiz-Torres, Paletta e M'Hallah (2017).

(a) Existe um conjunto de n tarefas independentes, $N = \{1, \ldots, j, \ldots, n\}$, para serem processadas em um conjunto de m máquinas paralelas, $M = \{1, \ldots, k, \ldots, m\}$.

(b) Todas as tarefas estão disponíveis para o processamento no tempo zero, são não-preemptivas, isto é, a execução de cada tarefa não pode ser interrompida até que seja totalmente concluída e cada tarefa deve ser processada uma única vez.

(c) Cada máquina pode processar somente uma tarefa de cada vez, não pode ficar ociosa até a última tarefa atribuída ter sido finalizada e estas máquinas são classificadas como máquinas paralelas não-relacionadas, isto é, cada tarefa j possui um tempo de processamento que depende da máquina k na qual será alocada.

(d) O tempo de processamento normal da tarefa j na máquina k é definido por p_{jk}.

(e) O efeito de deterioração da tarefa j na máquina k é dado por d_{jk}, em que, $0 \leq d_{jk} \leq 1$ $\forall j \in N$ e $k \in M$.

(f) O desempenho da máquina k na posição h (isto é, após executar a tarefa que está na posição $h-1$) é determinado por:

$$q_{k[h]} = (1 - d_{[h-1]k}) \times q_{k[h-1]} \tag{3.14}$$

em que, $d_{[h-1]k}$ é o efeito de deterioração provocado na máquina k pela execução da tarefa que está na posição $h-1$.

(g) Assume-se que todas as máquinas iniciam sem efeito de deterioração, sendo assim, o nível de desempenho para processar a tarefa na posição 1 é 100%, ou seja, $q_{k[1]} = 1, \forall k \in M$.

(h) O tempo de processamento atual de uma tarefa j processada na máquina k na posição h é determinado por:

$$p'_{jk} = \frac{p_{jk}}{q_{k[h]}} \tag{3.15}$$

(i) Existem n possíveis posições em cada máquina, e A é definido como o conjunto de posições.

(j) Cada tarefa j possui uma data de entrega e_j.

(k) A duração de uma tarefa de manutenção que retorna a máquina k ao seu estado inicial de desempenho é definida por u_k.

(l) Uma manutenção não pode interromper o processamento de uma tarefa que está em execução em uma máquina.

(m) Uma máquina está ocupada processando uma tarefa ou está em manutenção (nenhum tempo ocioso inserido é permitido).

(n) Cada máquina pode sofrer várias manutenções ao longo do sequenciamento. As operações de manutenção podem ocorrer simultaneamente em qualquer número de máquinas.

(o) A soma dos tempos de processamento reais para as tarefas atribuídas à máquina e a duração das operações de manutenção é definida por C_k.

3.3.2 Modelo Matemático

As variáveis de decisão usadas no modelo matemático são:

- C_k: tempo de processamento final da máquina k (soma dos tempos de processamentos das tarefas e das manutenções).
- C_{max}: máximo tempo de processamento de todas as máquinas (*makespan*), isto é., $C_{max} = max_{k \in M} C_k$.
- t_{kh}: tempo de atraso da tarefa atribuída à máquina k na posição h.
- q_{kh}: desempenho da máquina k na posição h.
- $x_{jkh} = \begin{cases} 1, \text{ se a tarefa } j \text{ é atribuída para a máquina } k, \text{ na posição } h \\ 0, \text{ caso contrário.} \end{cases}$
- $s_{kh} = \begin{cases} 1, \text{ se existe uma tarefa de manutenção na máquina } k, \text{ na posição } h \\ 0, \text{ caso contrário.} \end{cases}$

Um novo modelo de programação não-linear inteiro misto biobjetivo do problema, baseado nos trabalhos de Ruiz-Torres, Paletta e M'Hallah (2017) e Santos et al. (2022a), é apresentado a seguir,

$$\min C_{max} \tag{3.16}$$

$$\min TDT = \sum_{h \in A} \sum_{k \in M} \sum_{j \in N} t_{kh} \times x_{jkh} \tag{3.17}$$

$$\sum_{j \in N} x_{jkh} + s_{kh} \leq 1, \forall h \in A, k \in M \tag{3.18}$$

$$\sum_{h \in A} \sum_{k \in M} x_{jkh} = 1, \forall j \in N \tag{3.19}$$

$$\sum_{j \in N} \sum_{h \in A} \frac{p_{jk}}{q_{kh}} \times x_{jkh} + \sum_{h \in A} u_k \times s_{kh} \leq C_{max}, \forall k \in M \tag{3.20}$$

$$x_{jkh} - \sum_{l \in N} x_{lk(h-1)} - s_{k(h-1)} \leq 0, \forall j \in N, k \in M, h \in A \backslash \{1\} \tag{3.21}$$

$$q_{kh} = \sum_{j \in N} (1 - d_{jk}) \times q_{k(h-1)} \times x_{jk(h-1)} + s_{k(h-1)}, \forall h \in A \backslash \{1\}, k \in M \tag{3.22}$$

$$q_{k1} = 1, \forall k \in M \tag{3.23}$$

$$t_{kh} \geq \sum_{j \in N} \sum_{l=1 \ldots h} \frac{p_{jk}}{q_{kl}} \times x_{jkl} - \sum_{j \in N} e_j \times x_{jkh}, \forall k \in M, h \in A \tag{3.24}$$

$$t_{kh} \geq 0, \forall k \in M, h \in A \tag{3.25}$$

$$x_{jkh} \in \{0, 1\}, \forall j \in N, k \in M, h \in A \tag{3.26}$$

$$s_{kh} \in \{0, 1\}, \forall k \in M, h \in A \tag{3.27}$$

No modelo, o primeiro objetivo é minimizar o C_{max} (makespan) definido na Equação (3.16) e o segundo objetivo é minimizar o tempo total de atraso das entregas das tarefas, em inglês, *total delay time* (TDT) (3.17). As restrições (3.18) garantem que cada posição em cada máquina pode ter no máximo uma tarefa ou uma manutenção atribuída. As restrições (3.19) garantem que cada tarefa deve ser atribuída para exatamente uma posição em uma máquina. As restrições (3.20) garantem que a carga total em cada máquina seja igual ou menor que C_{max}. As restrições (3.21) garantem atribuições contínuas de tarefas. A equação (3.22) define o nível de desempenho de cada máquina para cada posição da tarefa, onde o nível de desempenho retorna ao valor um quando há uma operação de manutenção. A equação (3.23) define o nível de desempenho inicial de cada máquina. Para a primeira posição, todas as máquinas têm desempenho igual a um, ou seja, 100% de produtividade. A equação (3.24) define o tempo de atraso da tarefa atribuída à máquina k na posição h. Se o valor de $t_{kh} = 0$, isso indica que a tarefa está dentro do prazo, enquanto $t_{kh} > 0$ indica que a tarefa naquela posição/máquina está atrasada. A equação (3.25) garante que o tempo de atraso das tarefas seja maior ou igual a zero. Por fim, as equações (3.26) e (3.27) definem o valor binário para as variáveis x_{jkh} e s_{kh}.

O modelo de programação matemática é não-linear por causa da expressão $\frac{x_{jkh}}{q_{kh}}$ nas restrições (3.20) e (3.24); por causa da expressão $q_{kh} \times x_{jkh}$ nas restrições (3.22) e por causa da expressão $t_{kh} \times x_{jkh}$ na segunda função objetivo (3.17). Conforme mencionado anteriormente na Seção 3.2.2, da mesma forma que não foi possível resolver o modelo anterior por meio da estratégia de linearização ou de um solver, também se deparou com a mesma limitação para a resolução do modelo em questão. Portanto, a validação desse modelo foi conduzida utilizando uma técnica de enumeração, conforme detalhado na Seção 5.3.4.

3.3.3 Exemplo Numérico

Para caracterizar uma instância do problema abordado, um estudo de caso contendo $n = 10$ tarefas e $m = 3$ máquinas é apresentado a seguir. A Tabela 3 fornece tempos de processamento (p_{jk}), efeitos de deterioração (d_{jk}) para cada tarefa j em cada máquina k; e as datas de entrega de cada tarefa j encontram-se na última coluna. A duração de uma manutenção u_k em cada máquina é dada por: $u_1 = 3$; $u_2 = 1$; e $u_3 = 2$.

A Figura 2 mostra dois possíveis sequenciamentos, nos quais as tarefas são atribuídas às máquinas. Cada posição representa: o nível de desempenho da máquina ao iniciar a execução da tarefa (acima), o tempo de processamento atual (p'_{jk}), o tempo de atraso da tarefa (t_{kh}) e o tempo total de processamento, no final de cada tarefa (abaixo). As tarefas representadas na cor verde são entregues dentro do prazo e as tarefas na cor vermelho são entregues com atraso.

Tabela 3 – Uma instância do problema de sequenciamento com deterioração da máquina dependente de sequência, com $n = 10$ e $m = 3$.

Tarefa	p_{j1}	p_{j2}	p_{j3}	d_{j1}	d_{j2}	d_{j3}	e_j
1	31,78	2,80	81,08	0,031	0,010	0,018	38,55
2	12,18	98,61	34,46	0,032	0,028	0,018	36,99
3	75,20	18,53	36,70	0,019	0,040	0,011	85,02
4	20,43	43,63	89,43	0,040	0,026	0,031	69,50
5	56,39	65,43	2,12	0,048	0,017	0,011	31,75
6	26,12	15,82	29,49	0,024	0,015	0,043	94,94
7	82,53	82,17	32,54	0,033	0,028	0,043	76,72
8	61,04	49,39	32,86	0,026	0,012	0,023	101,72
9	98,26	86,35	56,47	0,045	0,023	0,017	126,53
10	93,46	97,21	73,49	0,024	0,049	0,033	119,70

A Figura 2 mostra o sequenciamento 1 com as tarefas 2, 4 e 8 atribuídas à máquina 1, as tarefas 3, 1 e 10 atribuídas à máquina 2 e as tarefas 6, 5, 7 e 9 atribuídas à máquina 3. Os valores da função objetivo são: *makespan* = 128,4 e TDT = 5,94. No sequenciamento 2, uma manutenção é escalonada na máquina 2 e uma manutenção é escalonada na máquina 3. Pode-se perceber que após a realização da manutenção, o desempenho da máquina volta a 100% naquela posição. Isso permite que as tarefas alocadas após a manutenção concluam o tempo de processamento dentro do prazo. Ou seja, a tarefa 10 atribuída à máquina 2 e a tarefa 9 atribuída à máquina 3 são processadas sem atraso na entrega. Assim, com o escalonamento de uma manutenção no sequenciamento 2, o valor do TDT diminuiu para 0,0 e o valor do *makespan* diminuiu para 124,55. Sendo assim, nota-se que o escalonamento da manutenção pode otimizar ambas as funções objetivo do problema.

3.4 Conclusão

Este capítulo abordou a problemática do sequenciamento de máquinas paralelas não-relacionadas, expandindo-a para um problema biobjetivo no qual o efeito da deterioração é influenciado pela sequência das tarefas nas máquinas. Adicionalmente, apresentou a integração das operações de manutenção no processo de sequenciamento, com o propósito de mitigar a deterioração das máquinas e, como resultado, reduzir o tempo de processamento final.

Neste contexto, foi formulado um modelo de programação matemática que visa a minimização simultânea do *makespan* e do tempo total de atraso nas entregas das tarefas. Posteriormente, um novo modelo foi formulado para considerar que as deteriorações das máquinas podem ser recuperadas por meio de operações de manutenção, permitindo a restauração do desempenho das máquinas.

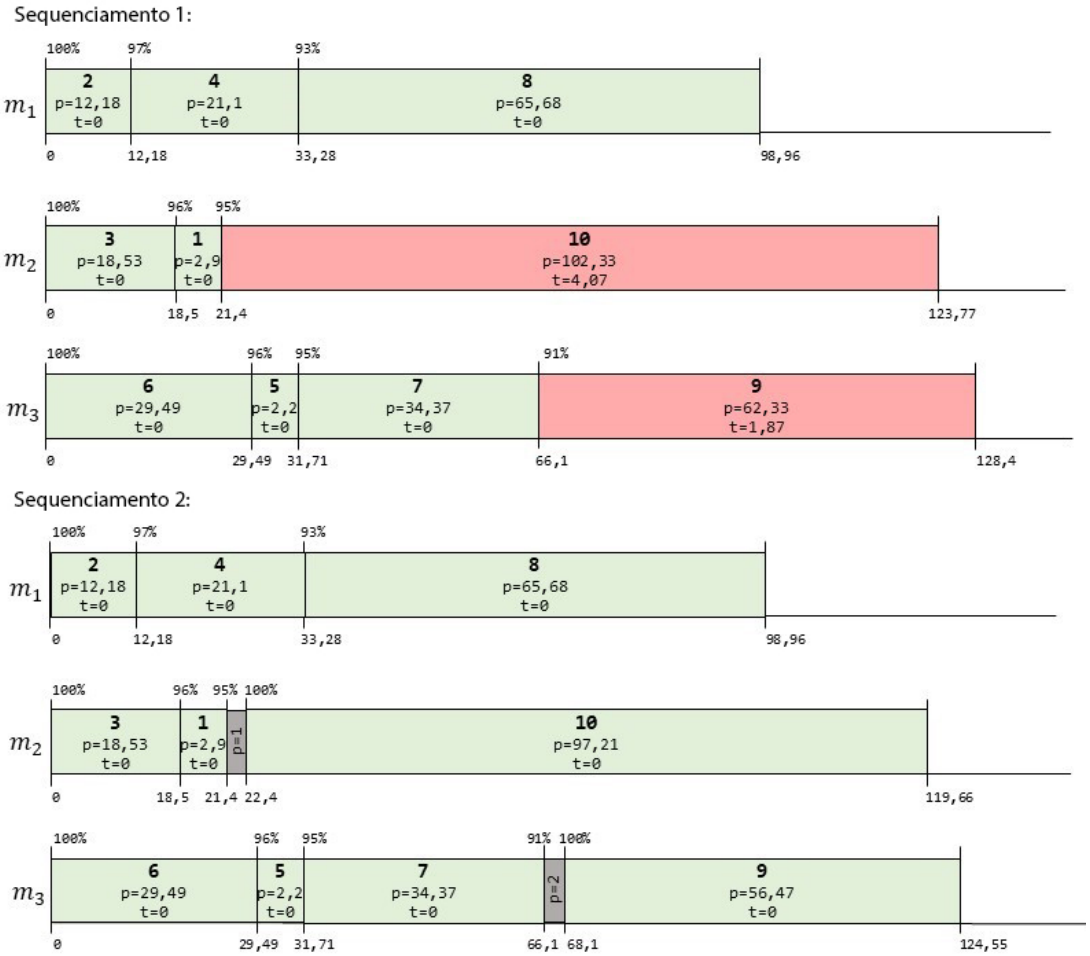


Figura 2 – Sequenciamentos para 10 tarefas e 3 máquinas.

Ao longo deste capítulo foram estabelecidos os fundamentos teóricos e conceituais necessários para a compreensão profunda do problema biobjetivo abordado, bem como para o desenvolvimento subsequente de algoritmos e estratégias de otimização. Os desafios envolvendo a interação entre sequenciamento e manutenção foram destacados, e a importância de encontrar soluções eficazes para esse problema foi enfatizada.

No capítulo seguinte serão apresentados os métodos de otimização multiobjetivo desenvolvidos para abordar esse problema complexo, abrindo caminho para uma análise mais detalhada e experimentos computacionais destinados a avaliar o desempenho das soluções propostas.

4 Algoritmos de Otimização Multiobejtivo

4.1 Introdução

A fronteira Pareto é um conceito central em otimização multiobjetivo. Ela define um conjunto de soluções não dominadas em um espaço de soluções, onde nenhuma solução pode ser melhorada em um objetivo sem piorar pelo menos um outro objetivo. Matematicamente, o conjunto de todas as soluções ótimas x é dito fronteira Pareto-ótimo se não existe um outro ponto y, tal que y é estritamente melhor que qualquer um dos pontos de x em todos os objetivos (MIETTINEN, 1999). Já a fronteira Pareto aproximada é um conjunto de soluções que se aproxima da fronteira Pareto-ótimo. Em muitos problemas práticos, é difícil ou impossível calcular a fronteira Pareto-ótimo devido à complexidade computacional ou à incerteza nos dados (COELLO, 2007).

Dada a complexidade de encontrar soluções que atendam simultaneamente a vários objetivos, meta-heurísticas têm sido empregadas como ferramentas de solução para os problemas multiobjetivo, buscando encontrar um conjunto de soluções que se aproxime da fronteira Pareto-ótimo (DEB, 2001). Dentre as várias categorias existentes, destacam-se as meta-heurísticas baseadas em população e meta-heurísticas baseadas em busca local.

As meta-heurísticas multiobjetivo baseadas em população utilizam a teoria da evolução para encontrar soluções de alta qualidade para problemas multiobjetivo. Estas meta-heurísticas operam em uma população de indivíduos, onde cada indivíduo é uma possível solução para o problema em questão. A população evolui ao longo de várias gerações, por meio da aplicação de operadores genéticos, como seleção, cruzamento e mutação, para melhorar a qualidade das soluções ao longo do tempo. Uma das vantagens dessas abordagens baseadas em população é sua capacidade de muitas vezes conseguir explorar uma ampla região do espaço de soluções e obter uma representação diversificada da fronteira Pareto (DEB, 2011).

Por outro lado, as meta-heurísticas multiobjetivo baseadas em busca local podem ser mais eficientes para problemas específicos ou quando os recursos computacionais são limitados. Estas abordagens partem do princípio de realizar ajustes locais em uma solução inicial (ou um pequeno conjunto de soluções) com o objetivo de melhorar sua qualidade. Os ajustes são geralmente orientados com base na qualidade dos objetivos e na vizinhança da solução atual, levando a uma convergência para um ótimo local (HANSEN; MLADENOVIĆ; PEREZ, 2008).

Neste contexto, as seções a seguir descrevem as meta-heurísticas utilizadas neste trabalho. Entre elas, duas são baseadas em população: *Multi-objective Evolutionary Algo-*

rithm based on Decomposition (MOEA/D) e *Non-Dominated Sorting Genetic Algorithm* (NSGAII), e uma é baseada em busca local: *Pareto Iterated Local Search* (PILS). Além disso, este capítulo apresenta um algoritmo híbrido proposto que utiliza as abordagens baseadas em população e busca local (Seção 4.5) e as estratégias para adicionar as operações de manutenção ao sequenciamento (Seção 4.6).

4.2 Non-Dominated Sorting Genetic Algorithm II (NSGA-II)

O algoritmo NSGA-II foi proposto por Deb et al. (2002), sendo considerado uma evolução do algoritmo NSGA original (SRINIVAS; DEB, 1994). O NSGA-II é amplamente reconhecido no campo da otimização multiobjetivo. Ele se destaca por sua capacidade de gerar soluções de alta qualidade para uma ampla variedade de problemas, oferecendo conjuntos de soluções que são bem distribuídos ao longo da fronteira Pareto aproximada.

O NSGA-II adota uma abordagem elitista, mantendo uma população de soluções de alta qualidade ao longo das gerações. Para fazer isso, o NSGA-II utiliza um procedimento de classificação que identifica indivíduos não dominados, chamado de *fast non-dominated sorting*. Neste procedimento, os indivíduos não dominados são colocados na fronteira Pareto que representam as melhores soluções encontradas até o momento. Estas soluções são diretamente transferidas para a próxima geração (DEB et al., 2002).

O NSGA-II utiliza um mecanismo para aumentar a diversidade das soluções encontradas, chamado *crowding distance* (distância de multidão). Este método melhora a distribuição das soluções nas fronteiras, evitando que as soluções fiquem próximas umas das outras e representando adequadamente a fronteira Pareto (DEB et al., 2002).

O Algoritmo 1 mostra o pseudocódigo simplificado do NSGA-II, apresentando apenas os passos básicos deste algoritmo. Este algoritmo inicia descrevendo os dados de entrada, P_0: a população inicial; s: tamanho da população e g: número máximo de gerações. A população é inicializada com s soluções (conhecida como população pai) e todos os indivíduos na população inicial são avaliados (linhas 1 e 2). Em seguida, inicia-se o laço de repetição principal do algoritmo que irá percorrer g gerações. Dentro deste laço (linhas 4 a 9), operadores genéticos (seleção, cruzamento e mutação) são aplicados à população atual P_t, gerando uma nova população Q_t, conhecida como população filha; as populações atual P_t e gerada Q_t são combinadas para formar uma população combinada R_t; uma ordenação baseada em dominância da população R_t é realizada, este mecanismo conhecido como *fast non-dominated sorting* divide a população R_t em diferentes fronteiras de dominância; a população para a próxima geração é inicializada como vazia e o contador para as fronteiras de dominância é inicializado. A partir de então, enquanto o tamanho da população da próxima geração mais o tamanho da próxima fronteira de dominância não exceder o tamanho máximo da população: calcula-se o *crowding distance* para cada solução

na fronteira F_i. Isso ajuda a manter a diversidade na população (linha 10); adiciona-se a fronteira F_i à população da próxima geração (linha 11); incrementa o contador de fronteiras (linha 12). Logo após, a fronteira F_i é ordenada usando o operador de comparação baseado no *crowding distance* (linha 14); as soluções da fronteira ordenada F'_i são adicionadas à população da próxima geração até que seu tamanho atinja s (linha 15). Ao final, o algoritmo retorna as soluções não dominadas da população da última geração (linha 17).

Algoritmo 1: NSGA-II

entrada : População inicial P_0, tamanho da população s, número de gerações g
saída : Fronteira Pareto aproximada

1 $P_0 \leftarrow$ inicializar População (s);
2 $P_0 \leftarrow$ avaliar as soluções de P_0;
3 **para** $t = 0$ *até* $t = g$ **faça**
4 $\quad Q_t \leftarrow$ aplicar Operadores Genéticos (P_t);
5 $\quad R_t \leftarrow P_t \cup Q_t$;
6 $\quad F \leftarrow$ ordene R_t usando *fast Non Dominated Sorting*;
7 $\quad P_{t+1} \leftarrow \emptyset$;
8 $\quad i \leftarrow 1$;
9 $\quad$ **enquanto** $|P_{t+1}| + |F_i| \leq s$ **faça**
10 $\quad\quad$ calcular *crowding Distance* (F_i);
11 $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup F_i$;
12 $\quad\quad i \leftarrow i + 1$;
13 $\quad$ **fim**
14 $\quad F'_i \leftarrow$ ordenar as soluções de F_i de acordo com *crowding Distance*;
15 $\quad P_{t+1} \leftarrow P_{t+1} \cup F'_i[1 : (s - |P_{t+1}|)]$;
16 **fim**
17 **retorna** Soluções Não Dominadas de P_g

4.3 Multi-objective Evolutionary Algorithm based on Decomposition (MOEA/D)

O algoritmo evolutivo multiobjetivo MOEA/D, proposto por Zhang e Li (2007), propõe a decomposição de problemas multiobjetivo em vários subproblemas de otimização escalar. Este algoritmo fundamenta-se em decompor o problema original multiobjetivo em subproblemas mais simples, utilizando vetores de peso. Cada vetor de peso representa uma direção no espaço dos objetivos. Desta forma, o algoritmo busca otimizar cada subproblema com base em sua direção associada, ao invés de procurar soluções não dominadas no espaço de objetivos original (LI; ZHANG, 2008).

As relações de vizinhança entre esses subproblemas são definidas com base nas distâncias entre seus vetores de peso de agregação. O subproblema i é considerado vizinho do subproblema j se o vetor de pesos do subproblema i for próximo ao do subproblema j. Cada subproblema é otimizado utilizando informações principalmente provenientes de seus subproblemas vizinhos (ZHOU et al., 2011).

Em cada iteração, o algoritmo tenta melhorar as soluções em direção a esses vetores de peso, utilizando informações da vizinhança de cada solução. Para isso, o algoritmo seleciona um subconjunto de soluções vizinhas e gera novas soluções por meio de operadores genéticos. Em seguida, o algoritmo atualiza a população considerando a solução atual, a nova solução gerada, e os vetores de peso associados (ZHANG; LI, 2007).

Uma vantagem significativa do MOEA/D é sua eficiência computacional. A decomposição do MOEA/D permite dividir um problema multiobjetivo em subproblemas menores e independentes, que podem ser resolvidos simultaneamente. Isso não apenas facilita a adaptação do algoritmo a diferentes contextos, mas também resulta em um menor custo computacional. No entanto, a escolha dos vetores de peso e o tamanho da vizinhança podem impactar significativamente a qualidade das soluções encontradas (DEB; JAIN, 2013).

O Algoritmo 2 apresenta o pseudocódigo básico do MOEA/D, fornecendo uma representação esquemática do processo geral do algoritmo. Este algoritmo inicia descrevendo os dados de entrada, s: tamanho da população; g: o número total de gerações; η: o tamanho da vizinhança B (quantidade de vizinhos a serem considerados); e W: os vetores de peso. Em seguida, os vetores de peso são inicializados. Esses vetores desempenham um papel crucial no MOEA/D, pois determinam a direção na qual cada subproblema é otimizado (linha 1); a população com s soluções é inicializada e os objetivos correspondentes são avaliados (linha 2); para cada vetor de peso w_i são identificados os η vizinhos mais próximos para compor vizinhança de B_i associada a w_i. Essa proximidade é geralmente determinada pela distância Euclidiana (linha 3). A partir de então, começam as repetições que representam o núcleo evolutivo deste algoritmo, onde as soluções são iterativamente melhoradas até que sejam exploradas g gerações (linhas 4 a 11). Para cada subproblema (linhas 5 a 10), seleciona-se aleatoriamente dois vizinhos da lista de vizinhos B_i; uma nova solução v é gerada pela recombinação (cruzamento e mutação) das soluções P_j e P_k; a nova solução v deve ser avaliada; as soluções na vizinhança de B_i são atualizadas. Se a nova solução v for melhor do que alguma solução existente na vizinhança do subproblema corrente, a solução existente é substituída por v. Ao final, o algoritmo retorna as soluções não dominadas da população (linha 12).

Neste trabalho, o algoritmo MOEA/D implementado foi ajustado para o problema em questão. Para isso, alguns ajustes foram feitos, tais como: a função *Tchebycheff* foi utilizada como função de agregação; o método *Scheffé* foi utilizado para inicializar os pesos W; o número máximo de atualizações feitas é definido por $\frac{\eta}{2}$ (conforme proposto por Trivedi et al. (2017)); os operadores de cruzamento, mutação e valores dos parâmetros utilizados são descritos no Capítulo 5.

Algoritmo 2: MOEA/D

entrada : Tamanho da população s, total de gerações g, tamanho da vizinhança η
saída : Soluções não dominadas

Inicializar vetores de peso $W = \{w_1, w_2, ..., w_s\}$;
Inicializar soluções $P = \{P_1, P_2, ..., P_s\}$ e avaliar funções objetivo;
Identificar os η vizinhos mais próximos para cada vetor de peso w_i baseado na distância Euclidiana para formar o conjunto B_i;
para $y = 1$ *até* $y = g$ **faça**
    para $i = 1$ *até* $i = s$ **faça**
        Escolher aleatoriamente dois vizinhos P_j e P_k da lista de vizinhos de B_i;
        Gerar v pela recombinação de P_j e P_k;
        Avaliar a solução v;
        Atualizar as soluções de P pertencentes a vizinhança de B_i ;
    fim
fim
retorna Soluções não dominadas de P;

4.4 Pareto Iterated Local Search (PILS)

Uma maneira natural de estender o ILS para problemas multiobjetivo é substituir a heurística de busca local por uma de busca local baseada em dominância Pareto. O PILS, também conhecido como ILS multiobjetivo, foi definido por Geiger (2006) e Geiger (2008). O algoritmo PILS é uma extensão direta da meta-heurística PLS. A meta-heurística PLS (PAQUETE; CHIARANDINI; STÜTZLE, 2004) define uma busca local baseada na dominância Pareto para a solução de problemas de otimização combinatória multiobjetivo. Neste trabalho, é desenvolvido um algoritmo PILS para o problema proposto a fim de contrastar esta abordagem com a estratégia de decomposição e agregação usada no algoritmo ILS/D.

O algoritmo PILS possui uma estrutura básica e poucos parâmetros, além disso, tem se mostrado uma abordagem de muito sucesso para resolver diversos problemas de sequenciamento. Geiger (2009) utilizou a metodologia do algoritmo PILS para a programação de *flow shop* multiobjetivo. Ele usou duas abordagens diferentes. Na primeira, dois objetivos foram considerados para minimizar: o *makespan* e o atraso total ponderado; enquanto na segunda, três objetivos foram estudados: *makespan*, o tempo médio de conclusão e o tempo médio de atraso. Da mesma forma, Geiger (2011) estudou os problemas de programação de permutação *flow shop* com múltiplos objetivos e comparou com outras abordagens de busca local. Xu e Lei (2014) estudaram um problema de programação de *flow shop* de dois agentes e propuseram o algoritmo PILS para minimizar o *makespan* das tarefas para o primeiro agente e o atraso total das tarefas para o segundo agente, simultaneamente. Gomes, Neves e Souza (2014) abordam o problema de sequenciamento de projetos com restrições de recursos com relações de precedência, e visam minimizar dois

critérios: o *makespan* e o tempo total ponderado de início das atividades. Para resolver o problema, eles usam o algoritmo PILS entre outras meta-heurísticas multiobjetivo.

O algoritmo PILS, desenvolvido neste trabalho, possui cinco etapas: (i) construção de soluções iniciais; (ii) seleção de uma solução do conjunto de soluções não dominadas para ser explorada; (iii) exploração da vizinhança; (iv) verificação de quais soluções já foram visitadas; e (v) aplicação da perturbação.

Algoritmo 3: PILS

entrada : tempo máximo de execução τ, quantidade de soluções iniciais ϕ, tamanho da perturbação t
saída : conjunto de soluções não dominadas ND

```
1  ND ← Procedimento Construtivo(φ);
2  Selecione uma solução p ∈ ND;
3  ϒ ← quantidade de estruturas de vizinhanças a serem exploradas;
4  enquanto (τ não satisfeito) faça
5      para i = 1 até i = ϒ faça
6          para p' ∈ E_i(p) faça
7              ND ← soluções não dominadas obtidas de ND ∪ {p'};
8          fim
9          se ( ∃ p' ∈ E_i(p) | p' domina p) então
10             p ← p';
11             Reordene as vizinhanças E_1, ..., E_ϒ aleatoriamente;
12             i ← 1;
13         fim
14     fim
15     Defina p como visitada;
16     se ( ∃ q ∈ ND | q não foi visitado ainda) então
17         p ← q;
18     fim
19     senão
20         Selecione uma solução p ∈ ND;
21         p ← Perturbação (p, t);
22     fim
23 fim
24 retorna ND;
```

Uma descrição geral do PILS é detalhada no Algoritmo 3. O algoritmo possui três parâmetros de entrada: τ (controla o tempo máximo de execução), ϕ (define a quantidade de soluções criadas pelo procedimento construtivo) e t (define o tamanho fixo da perturbação). Na linha 1, o procedimento construtivo cria ϕ soluções iniciais. Essas soluções formam o conjunto de soluções não dominadas (ND). Em seguida, é selecionada, aleatoriamente, uma das soluções no conjunto ND para torna-se a solução atual p (linha 2) e definida a quantidade Υ de estruturas de vizinhanças a serem exploradas (linha 3).

As iterações do algoritmo PILS ocorrem no intervalo entre as linhas 4 e 23,

enquanto o critério de parada (τ) não é satisfeito. Durante cada iteração, até que Υ estruturas de vizinhança ($E_1, ..., E_i, ..., E_\Upsilon$) sejam executadas, explora-se toda a vizinhança de p e atualiza-se o conjunto ND (linhas 6-8). Em seguida, verifica-se em toda a vizinhança E_i, se alguma solução vizinha p' domina a solução corrente p (linha 9). Se essa condição for verdadeira, a melhor solução p' torna-se a nova solução atual (linha 10); reordenam-se as vizinhanças aleatoriamente (linha 11) e retorna-se à primeira vizinhança na nova ordem gerada (linha 12). Caso contrário, o algoritmo avança para a próxima vizinhança. Ao concluir, a solução p é marcada como visitada (linha 15), e é verificado se existem soluções não visitadas no conjunto ND (linha 16). Se essa condição for verdadeira, uma solução não visitada é selecionada aleatoriamente para se tornar a solução p atual (linha 17). Caso contrário, uma solução do conjunto ND é selecionada aleatoriamente (linha 20), e um método de perturbação é aplicado (linha 21). Em seguida, a solução p torna-se a nova solução atual. Finalmente, o conjunto de soluções ND é retornado (linha 24).

O procedimento construtivo, o procedimento de perturbação e as estruturas de vizinhanças usadas pelo algoritmo PILS são os mesmos do algoritmo ILS/D, os quais são explicados na Seção 4.5.1, Seção 4.5.2.1 e Seção 4.5.2.2, respectivamente.

4.5 Iterated Local Search Based on Decomposition (ILS/D)

Para resolver o problema, é proposto um algoritmo heurístico baseado na abordagem de decomposição e agregação do algoritmo MOEA/D. O algoritmo MOEA/D decompõe um problema de otimização multiobjetivo em vários subproblemas de otimização escalar e os otimiza simultaneamente com uma população de soluções candidatas. Cada subproblema é otimizado usando apenas informações de seus vários subproblemas vizinhos, o que faz com que o MOEA/D tenha menor complexidade computacional (ZHANG; LI, 2007). Neste novo método, a meta-heurística ILS é usada como mecanismo de busca. Ou seja, um ILS mono-objetivo é usado para otimizar cada subproblema. O novo algoritmo híbrido é denominado *Iterated Local Search Based on Decomposition* (ILS/D). Uma vantagem do ILS/D é que, devido à estrutura de decomposição e agregação não é necessário o uso de uma busca local baseada em dominância Pareto. Além disso, os vetores de pesos podem permitir uma melhor aproximação da fronteira Pareto e controle da diversidade.

De forma semelhante ao MOEA/D, o algoritmo proposto ILS/D emprega a abordagem de decomposição e agregação, fazendo a setorização da busca na aproximação da fronteira Pareto. Sendo assim, o ILS/D divide um problema multiobjetivo em problemas mono-objetivo com base em agregações usando diferentes vetores de peso.

O algoritmo ILS/D possui a estrutura do MOEA/D clássico, mas usa o algoritmo ILS (LOURENÇO; MARTIN; STÜTZLE, 2003) para resolver os subproblemas escalares, ou seja, o ILS é usado para melhorar o processo de busca dentro da estrutura do MOEA/D.

A principal vantagem dessa abordagem é que o ILS pode ser empregado com uma busca local mono-objetivo e estendido diretamente para problemas multiobjetivo.

O método *Scheffé* (SCHEFFÉ, 1958) é usado para construir um conjunto de pesos w para todos os subproblemas. Neste método, uma teoria é desenvolvida para experimentos com misturas de componentes, cujo objetivo é a previsão empírica da resposta a qualquer mistura dos componentes, quando a resposta depende apenas da proporção dos componentes e não da quantidade total. Este método é definido pela proposição $\{R, w_{max}\}$-*simplex lattice*. Tem-se que R é o número de objetivos em problemas de otimização. No método de *Scheffé*, são gerados $(R + w_{max} - 1, w_{max})$ pontos no espaço R-dimensional, com $w_{max} + 1$ pontos igualmente espaçados no limite e satisfazendo a condição: $\|w\|_1 = w_1 + w_2 + ... + w_R = 1$ (COTA et al., 2019).

O pseudocódigo do ILS/D é mostrado no Algoritmo 4. Os parâmetros de entrada do Algoritmo 4 podem ser divididos entre parâmetros do ILS/D e parâmetros do ILS. Os parâmetros do ILS/D são descritos a seguir, s: total de subproblemas gerados por decomposição; η: número de vizinhos de cada subproblema[1]; e g: número máximo de gerações. Os parâmetros τ, t_{min} e t_{max} são utilizados pelo ILS e são descritos na Seção 4.5.2.

Na linha 1 do Algoritmo 4, um conjunto de pesos, denominados W, é construído pelo método *Scheffé* para o número s de subproblemas. Uma população, P, é construída como um conjunto de indivíduos (P_i). Na linha 2, P é construído com s indivíduos gerados pelo procedimento construtivo. Cada indivíduo na população (P_i) representa uma solução candidata para o subproblema correspondente (i) e está associado a um vetor de peso (w_i). Na linha 5, o conjunto de η vizinhos de cada subproblema i é definido para B_i, que contém η vetores de pesos mais próximos de w_i, $v \in B_i$. Para definir esses subproblemas vizinhos, é calculada a distância Euclidiana entre w_i de P_i e todos os outros pesos do conjunto W. Assim, aqueles η vizinhos que têm o menor valor de distância são definidos como o conjunto B_i.

A estimativa do ponto ideal no espaço objetivo é denotada como z^*. Este é definido pelo ponto formado pelo menor valor de *makespan* e o menor valor de TDT encontrado durante a execução do algoritmo até o momento, ou seja, z^* é atualizado com a melhor solução para cada objetivo. Nas linhas 7 e 8, z^* é inicializado e atualizado, respectivamente.

Na linha 9, o laço mais externo que controla cada geração, até um máximo de g gerações é definido. Na linha 10, é definido o laço interno que controla cada um dos

[1] No MOEA/D, e também no ILS/D, cada vetor de peso define um subproblema com a agregação dos objetivos. Os vetores de peso próximos um dos outros no espaço de busca representam subproblemas vizinhos. Portanto, este conceito de vizinhança de vetores de peso e subproblemas correspondentes no espaço objetivo não deve ser confundido com o conceito mais usual de estruturas de vizinhança usado em heurísticas de busca local.

subproblemas i até o número total s. Nas linhas 11 a 13, o vizinho[2] v do sub-problema i é selecionado aleatoriamente a partir de B_i. A solução correspondente associada a v é passada como parâmetro ao algoritmo ILS, tornando-se a solução inicial, e P_i é atualizado com a solução otimizada por ILS. Em seguida, o vetor das melhores soluções z^* é atualizado. Nas linhas 14 a 18, usando a função de agregação *Tchebycheff*, verifica-se se a solução P_i é melhor do que a solução P_j, para cada vizinho $w_j \in B_i$. Se verdadeiro, P_j é atualizado com a solução P_i. Neste caso, o número máximo de atualizações que podem ser feitas é definido por $\frac{\eta}{2}$, conforme recomendado na literatura (TRIVEDI et al., 2017). Por fim, as soluções que são dominadas na população P são removidas e o conjunto P restante é retornado (linha 22). O comportamento do Algoritmo 4 é visualmente ilustrado no Apêndice A.

Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}

saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *até* $i = s$ **faça**
4 Associe P_i com w_i;
5 $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ *até* $y = g$ **faça**
10 **para** $i = 1$ *até* $i = s$ **faça**
11 Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 $P_i \leftarrow$ ILS(τ, t_{min}, t_{max}, p, w_i, z^*);
13 Atualize z^* considerando os valores dos objetivos de P_i;
14 **para** *cada vizinho* $w_j \in B_i$ **faça**
15 **se** $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 $P_j \leftarrow P_i$;
17 **fim**
18 **fim**
19 **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

É importante destacar que para verificar se a solução P_i é melhor do que a solução P_j, utiliza-se a função de agregação *Tchebycheff*, baseada em Trivedi et al. (2017). O

[2] Novamente, observe que neste caso, um vizinho se refere a outro vetor de peso vizinho $v \in B_i$ na vizinhança do vetor de peso atual w_i.

subproblema i é definido pela minimização da seguinte função agregada:

$$f^{te}(x|w_i, z^*) = \max_{1 \leqslant r \leqslant R} \{w_i^r | f_r(x) - z_r^*|\} \tag{4.1}$$

em que, $z^* = (z_1^*, ..., z_R^*)$ é o ponto de referência ideal com: $z_r^* \leq \min\{f_r(x)|x \in \Omega\}$ para $r = 1, 2, ..., R$, onde Ω é o espaço de busca e x é o vetor de variável de decisão. Portanto, é importante mencionar que o algoritmo ILS na linha 12 do Algoritmo 4 é aplicado a um subproblema escalar representado pela função de agregação *Tchebycheff* (4.1) usando w_i e uma boa solução $p \in P$ de um subproblema vizinho $v \in B_i$ como a solução inicial.

A complexidade computacional do algoritmo MOEA/D é dada por $\mathcal{O}(Rs\eta)$, onde R é o número de objetivos, s é o tamanho da população e η é o número de vizinhos (ZHANG; LI, 2007). A complexidade computacional do algoritmo ILS abordado neste trabalho é dada por $\mathcal{O}(\tau)$, onde τ é uma constante em função do tempo de execução, que é definido por $0,08 \times \frac{n}{m}$ (conforme apresentado na Seção 5.2.2). Pode-se concluir que a complexidade computacional do algoritmo ILS/D é dada por $\mathcal{O}(Rs\eta\tau)$ para cada geração.

O procedimento construtivo é descrito na Seção 4.5.1 e o algoritmo ILS é descrito na Seção 4.5.2. Em seguida, é descrita a busca local (Seção 4.5.2.2). Neste procedimento, a avaliação do critério de aceitação é feita pela função de agregação *Tchebycheff*.

4.5.1 Procedimento Construtivo

O procedimento construtivo é usado para gerar indivíduos para a população inicial (P). Cada indivíduo da população (P_i) representa uma solução candidata para o subproblema i e está associado a um peso w_i. Inicialmente, divide-se o número de subproblemas s em três grupos de mesmo tamanho ($s/3$). No primeiro grupo, são gerados os indivíduos que têm o menor tempo de conclusão do processamento das tarefas (*makespan*), verificando a melhor posição e a melhor máquina para atribuir uma tarefa. No segundo grupo, são gerados os indivíduos que possuem um TDT menor para entrega de tarefas, verificando a melhor posição e a melhor máquina para atribuir uma tarefa. No terceiro grupo, os indivíduos são gerados aleatoriamente, ou seja, a posição e a máquina para atribuir uma tarefa são escolhidas de forma aleatória. Essa divisão em três grupos foi necessária para ter um conjunto diversificado de soluções iniciais na geração de indivíduos na população, uma vez que leva em consideração as duas funções objetivo do problema e também algumas soluções aleatórias.

O procedimento construtivo é apresentado no Algoritmo 5. Primeiramente, o conjunto P é inicializado, então, inicia-se o laço para definir como a solução será construída de acordo com cada um dos três grupos explicados anteriormente. Em seguida, inicia-se

o laço para criar $s/3$ soluções pertencentes a um grupo. Na linha 4, obtém-se uma *lista* composta por n tarefas (ordenada aleatoriamente) que irão compor uma solução p. Para cada tarefa j da *lista* (linha 6), procura-se a melhor máquina k para inserir j de acordo com o grupo corrente (linha 7). Em seguida, encontra-se a melhor posição h na máquina k para a tarefa ser atribuída (linha 8). Na linha 9, a tarefa j é inserida na posição h da máquina k da solução p que está sendo construída. Na linha 11, a solução p construída é armazenada no conjunto P. O algoritmo finaliza quando s soluções são criadas, as quais formam o conjunto P de soluções iniciais.

Algoritmo 5: Procedimento Construtivo

```
   entrada : total de subproblemas s
   saída   : população P
 1 P ← ∅
 2 para cada grupo faça
 3     para i = 1 até (s/3) faça
 4         lista ← n tarefas;
 5         Seja p a solução a ser construída;
 6         para j ∈ lista faça
 7             k ← melhor máquina para j de acordo com o grupo;
 8             h ← melhor posição para j em k;
 9             p ← tarefa j na posição h da máquina k;
10         fim
11         P ← P ∪ {p}
12     fim
13 fim
14 retorna P;
```

4.5.2 Algoritmo ILS

O algoritmo ILS, proposto por Lourenço, Martin e Stützle (2003), é uma meta-heurística simples e de aplicação geral que usa, iterativamente, um procedimento de perturbação como um mecanismo de diversificação, e uma busca local como uma heurística de melhoria. A cada iteração, uma nova solução é gerada realizando aleatoriamente uma modificação apropriada, chamada de perturbação, para uma solução ótima local encontrada anteriormente (solução atual). Em vez de gerar uma nova solução inicial desde o início, a perturbação gera uma solução promissora e tendenciosa ao reter parte da estrutura que tornou a solução atual boa. A solução perturbada é então melhorada pela heurística de busca local, obtendo uma nova solução. Esta nova solução é aceita como a nova solução atual sob algumas condições. Uma explicação detalhada do ILS pode ser encontrada em Lourenço, Martin e Stützle (2010).

O algoritmo ILS provou ser uma abordagem de muito sucesso para resolver diferentes problemas de otimização combinatória, como problemas da produção (CROCE;

GARAIX; GROSSO, 2012), (RIBAS; COMPANYS; TORT-MARTORELL, 2013); o problema de atribuição quadrática (STÜTZLE, 2006) e problemas de roteamento de veículos (PENNA; SUBRAMANIAN; OCHI, 2013), entre outros.

Neste trabalho, não é necessário um método para gerar a solução inicial para o algoritmo ILS, pois uma solução inicial é passada como parâmetro pelo algoritmo ILS/D, (ver na linha 12 do Algoritmo 4). Além disso, a função de avaliação usada pelo ILS é substituída pela Equação 4.1. Esta função é aplicada para agregar as duas funções objetivo, C_{max} e TDT, em um problema de objetivo único.

Uma descrição geral do pseudocódigo do ILS é mostrada no Algoritmo 6. O algoritmo tem seis parâmetros de entrada: τ (controla o tempo máximo de execução), t_{min} (define o tamanho mínimo de perturbação), t_{max} (define o tamanho máximo de perturbação), p (armazena a solução inicial do algoritmo ILS), w (contém o vetor de peso, usado pela função de agregação) e z^* (contém o ponto formado com a melhor solução para cada objetivo, usado pela função de agregação). Na linha 1, a solução inicial passada como parâmetro pelo algoritmo ILS/D é refinada pelo procedimento de busca local. Na linha 2, a melhor solução p^* é inicializada com a solução retornada da busca local. Na linha 3, o tamanho da perturbação t é inicializado com o valor mínimo t_{min}.

As principais iterações do algoritmo ILS são calculadas entre as linhas 4 e 20, até que o critério de parada seja satisfeito. Durante cada iteração, a solução atual p é perturbada (linha 5) e melhorada pelo procedimento de busca local obtendo uma nova solução p' (linha 6). Nas linhas 7 a 10, se a solução p' melhorar a melhor solução obtida até o momento, o tamanho da perturbação é definido para seu tamanho mínimo ($t = t_{min}$). Se a melhor solução não for melhorada, o tamanho da perturbação é incrementado (linhas 11 a 16). O valor máximo para o tamanho da perturbação é t_{max}. Se o tamanho da perturbação exceder este limite, seu valor é redefinido para o valor mínimo ($t = t_{min}$). Nas linhas 17 a 19, o critério de aceitação é testado. Se p' é melhor que a solução atual p, então, p' é aceita (p' substitui p). A melhor solução encontrada por todas as iterações é retornada pelo algoritmo (linha 21).

As próximas seções fornecem uma explicação detalhada dos principais componentes usados no algoritmo ILS.

4.5.2.1 Procedimento de Perturbação

O procedimento de perturbação é baseado em uma realocação circular de tarefas entre máquinas distintas, conhecido como *ejection chain* (GLOVER, 1996). Primeiro, um conjunto de t máquinas é escolhido aleatoriamente: $\{m_1, m_2, ..., m_t\}$. Este conjunto deve conter a máquina que possui o *makespan*. Em seguida, uma tarefa é selecionada aleatoriamente em cada máquina. A tarefa da máquina m_1 é inserida em uma posição

Algoritmo 6: ILS

entrada : tempo máximo de execução τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}, solução inicial p, vetor de peso w, ponto formado com a melhor solução para cada objetivo z^*
saída : melhor solução encontrada $p*$

```
1  p ← busca local(p, w, z*);
2  p* ← p;
3  t ← t_min;
4  enquanto (τ não satisfeito) faça
5  |  p' ← perturbação(p, t);
6  |  p' ← busca local(p', w, z*);
7  |  se f^te(p'|w, z*) ≤ f^te(p*|w, z*) então
8  |  |  p* ← p';
9  |  |  t ← t_min;
10 |  fim
11 |  senão
12 |  |  t ← t + 1;
13 |  |  se (t > t_max) então
14 |  |  |  t ← t_min;
15 |  |  fim
16 |  fim
17 |  se f^te(p'|w, z*) ≤ f^te(p|w, z*) então
18 |  |  p ← p';
19 |  fim
20 fim
21 retorna p*;
```

aleatória da máquina m_2, a tarefa da máquina m_2 é inserida em uma posição aleatória da máquina m_3, assim por diante até que a tarefa da máquina m_t seja inserida na máquina m_1.

O parâmetro t define o tamanho da perturbação. No algoritmo ILS proposto neste trabalho, o tamanho da perturbação é modificado durante o processo de busca, dentro do intervalo $[t_{min}, t_{max}]$, assim como foi abordado por Santos (2016). Nesse contexto, t é inicializado com o valor mínimo t_{min}. Se a melhor solução atual não for melhorada, t é incrementado por um. Se esta melhor solução é melhorada ou o tamanho da perturbação excede o valor máximo, t é reiniciado com o valor mínimo.

A justificativa para um valor crescente de t na perturbação é explorar soluções progressivamente mais distantes (da atual) por meio da busca local. Sempre que uma transição para uma nova solução é aceita, o tamanho da perturbação será mínimo novamente ($t = t_{\min}$) causando pequenas mudanças na nova solução situada em uma nova região. Retornar o valor de t para $t_{\min}$ quando t excede $t_{\max}$ é para evitar que o tamanho da perturbação permaneça com o valor muito alto por um longo tempo e gere soluções muito aleatórias.

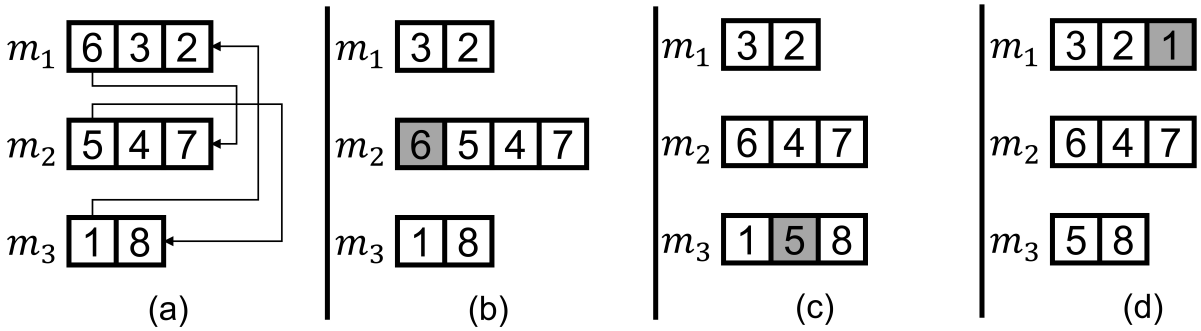


Figura 3 – Exemplo de perturbação para um sequenciamento com três maquinas.

A Figura 3 mostra o movimento de perturbação para $M = \{m_1, m_2, m_3\}$. Na primeira solução, apresentada na Figura 3(a), as realocações que são feitas entre as máquinas são ilustradas. Na Figura 3(b), a tarefa 6 é removida da máquina 1 e inserida na máquina 2. Na Figura 3(c), a tarefa 5 é removida da máquina 2 e inserida na máquina 3. Por fim, na Figura 3(d), a tarefa 1 é removida da máquina 3 e inserida na máquina 1.

4.5.2.2 Busca Local

O método de busca local é usado para melhorar a solução inicial e as soluções obtidas pelo procedimento de perturbação. A busca local é um método que inicia com uma solução (solução atual), e move para um vizinho melhor na vizinhança[3] da solução. A vizinhança contém todas as soluções alcançadas por meio de movimentos simples feitos na solução atual. No algoritmo ILS, a busca local é realizada por um procedimento baseado em *Variable Neighborhood Descent* (VND) (MLADENOVIĆ; HANSEN, 1997), que utiliza uma ordenação de vizinhança aleatória, denominado *Randon Variable Neighborhood Descent* (RVND), ou seja, as vizinhanças são exploradas em uma sequência aleatória (SUBRAMANIAN et al., 2010; SOUZA et al., 2010). O método de busca local é apresentado no pseudocódigo do Algoritmo 7.

As seis (valor definido para Υ) estruturas de vizinhanças utilizadas neste trabalho podem ser divididas em dois grupos, entre máquinas diferentes e na mesma máquina, que são descritas a seguir.

4.5.2.2.1 Estruturas de Vizinhança Entre Máquinas Diferentes

Quatro tipos de estruturas de vizinhança foram implementados entre máquinas diferentes, três delas exploram soluções usando o objetivo *makespan*, enquanto que a última usa o objetivo TDT. No entanto, note que, para uma determinada solução, essas estruturas podem afetar a valorização de ambos os objetivos. Essas estruturas são definidas a baixo:

[3] A palavra vizinho se refere ao conceito de estrutura de vizinhança no procedimento de busca local.

Algoritmo 7: busca local

entrada : tempo máximo de execução τ, solução inicial p', vetor de peso w, ponto formado com a melhor solução para cada objetivo z^*
saída : melhor solução encontrada $p*$

```
1  p* ← p';
2  ϒ ← quantidade de estruturas de vizinhanças a serem exploradas;
3  L ← gere um conjunto com o índice de cada estrutura de vizinhança {1, ..., ϒ};
4  enquanto (L ≠ ∅ ou τ não satisfeito) faça
5  |  i ← selecione aleatoriamente um índice em L correspondente a uma estrutura
   |    de vizinhança;
6  |  P ← gere as soluções vizinhas de p' com a estrutura de vizinhança E_i;
7  |  para p ∈ P faça
8  |  |  se f^te(p|w, z*) < f^te(p'|w, z*) então
9  |  |  |  p' ← p;
10 |  |  fim
11 |  fim
12 |  se f^te(p'|w, z*) < f^te(p*|w, z*) então
13 |  |  p* ← p' e L ← {1, ..., ϒ};
14 |  fim
15 |  senão
16 |  |  L ← L/{i};
17 |  fim
18 fim
19 retorna p*;
```

- O movimento de *troca par a par* (E_1)consiste em trocar duas tarefas j_1 e j_2 pertencentes a duas máquinas diferentes m_c e m_k, respectivamente, de tal forma que a primeira máquina m_c é a máquina com um tempo máximo de conclusão (máquina *makespan*). Se a máquina *makespan* tiver n_s tarefas, $n_s \times (n - n_s)$ soluções vizinhas podem ser obtidas pelo movimento de *troca par a par*, onde n é o número total de tarefas.

- O movimento *inserção única* (E_2) consiste em remover uma tarefa j da máquina *makespan* m_c e inseri-la em outra máquina m_k. Com este movimento, $n_s \times (m - 1)$ soluções vizinhas podem ser obtidas, onde m é o número de máquinas.

- O movimento *inserção dupla* (E_3) consiste em remover duas tarefas j_1 e j_2 da máquina *makespan* m_c e inseri-las em outra máquina m_k. Com este movimento, $n_s \times (m - 1) \times 2$ soluções vizinhas podem ser obtidas.

- O movimento *troca par a par considerando atraso* (E_4) consiste em trocar duas tarefas j_1 e j_2 pertencentes a duas máquinas diferentes m_d e m_k, respectivamente, de tal forma que a máquina m_d possui o máximo tempo de atraso nas entregas das tarefas. Se a máquina m_d tiver n_s tarefas, $n_s \times (n - n_s)$ soluções vizinhas podem ser obtidas.

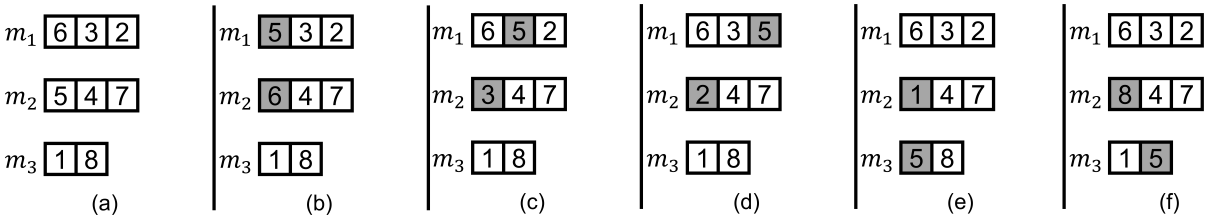


Figura 4 – Exemplo da estrutura de vizinhança *troca par a par* para um sequenciamento com três máquinas.

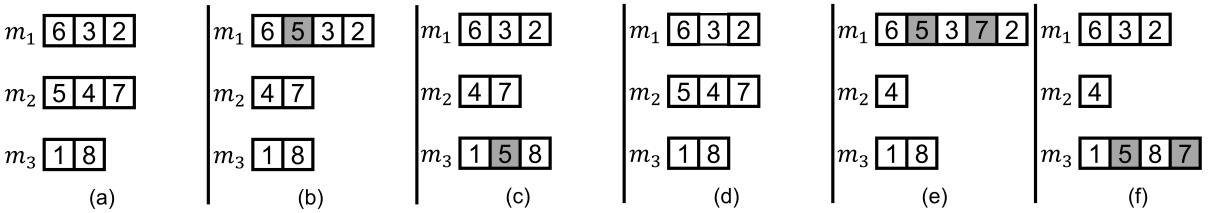


Figura 5 – Exemplo da estrutura de vizinhança *inserção única* e *inserção dupla* para um sequenciamento com três máquinas.

Para melhor entendimento dessas estruturas de vizinhanças, a Figura 4, descreve a operação da *troca par a par*. Considere a Figura 4 (a) como a solução inicial. O movimento de troca começa a partir da tarefa 5 da máquina que possui *makespan*, neste caso, a máquina 2. Em seguida, é realizada a primeira iteração dos movimentos: a troca da tarefa 5 com todas as outras tarefas das máquinas restantes. Na Figura 4 (b)-(c)-(d), a tarefa 5 da máquina 2 é trocada com a tarefa 6, 3 e 2 da máquina 1, respectivamente. Na Figura 4 (e)-(f), a tarefa 5 da máquina 2 é trocada com a tarefa 1 e a tarefa 8 da máquina 3, respectivamente.

A Figura 5 mostra um exemplo de iteração das estruturas de vizinhança *inserção única* e *inserção dupla*. Considere a Figura 5 (a) como solução inicial para a estrutura *inserção única*. Na Figura 5 (b)-(c), a tarefa 5 é removida da máquina 2 e inserida na máquina 1 e na máquina 3, respectivamente. A Figura 5 (d) apresenta um exemplo de iteração da estrutura de vizinhança *inserção dupla*. Na Figura 5 (e)-(f), as tarefas 5 e 7 são removidas da máquina 2 e inseridas nas máquinas 1 e 3, respectivamente.

4.5.2.2.2 Estruturas de Vizinhança na Mesma Máquina

O principal objetivo das estruturas de vizinhança na mesma máquina é reduzir o TDT para entregas das tarefas, obtendo assim mais entregas dentro do prazo. Essas estruturas são definidas a seguir:

- O movimento *troca* (E_5) consiste em trocar duas tarefas j_1 e j_2 na mesma máquina m_k.

- O movimento *inserção* (E_6) consiste em remover uma tarefa j da máquina m_k e inseri-la na mesma máquina m_k, em outra posição.

4.6 Estratégias para Sequenciamento das Manutenções

Esta seção foca em propor três diferentes estratégias para alocar as tarefas de manutenção no sequenciamento, usando o algoritmo híbrido ILS/D. Este algoritmo foi selecionado porque, como é mostrado no Capítulo 5, obtém desempenho melhor do que outros algoritmos multiobjetivo, como MOEA/D, NSGA-II e PILS. A primeira estratégia para alocar manutenção não considera as tarefas de manutenção em toda a execução do algoritmo ILS. Ou seja, o comportamento original do ILS e o tempo de execução do algoritmo ILS não foram comprometidos. A segunda estratégia de manutenção não considera as tarefas de manutenção nos procedimentos de perturbação e busca local do algoritmo ILS. Contudo, esta estratégia considera as tarefas de manutenção nas soluções após a vizinhança ser gerada pela busca local. Ou seja, uma parte do algoritmo ILS foi alterado. A terceira estratégia de manutenção considera as tarefas de manutenção em todo processamento do algoritmo ILS. Ou seja, o algoritmo ILS foi inteiramente alterado para processar as tarefas de manutenções da mesma forma que as outras tarefas da máquina. Estas três estratégias são desenvolvidas e os seus resultados são comparados entre si.

Neste trabalho, para verificar quando uma manutenção é necessária em uma solução é utilizado o Lema 1 definido por Ruiz-Torres, Paletta e M'Hallah (2017). O Lema 1 define que uma manutenção deve ser adicionada (*scheduled*) para reduzir o tempo de processamento final em uma máquina quando a diferença entre o tempo de processamento real (p'_{jk}) e o tempo de processamento inicial (p_{jk}) de uma determinada tarefa j em uma máquina k for maior que o tempo gasto para manutenção naquela máquina k (u_k). A seguir é apresentado de forma sucinta o Lema 1 definido por Ruiz-Torres, Paletta e M'Hallah (2017). O detalhamento e prova do Lema 1 encontra-se em Ruiz-Torres, Paletta e M'Hallah (2017).

Lema 1: se $p'_{jk} - p_{jk} > u_k$, então, uma tarefa de manutenção deve ser adicionada imediatamente antes desta tarefa j para diminuir a função objetivo.

O tempo de processamento para uma tarefa subsequente pode ser significativamente aumentado pela deterioração da máquina causada pelo processamento das tarefas anteriores. O Lema 1 definido por Ruiz-Torres, Paletta e M'Hallah (2017) afirma que a manutenção deve ser adicionada se o tempo de processamento adicional causado pela deterioração na máquina exceder o tempo gasto para realizar a manutenção. Ao incluir a tarefa de manutenção, o desempenho da máquina será restaurado para 100%, o que reduzirá os tempos de processamento das tarefas subsequentes, reduzindo assim o tempo de processamento final da máquina.

O algoritmo ILS/D (Algoritmo 4) foi ajustado para adicionar as tarefas de manutenção em cada máquina, de acordo com o estabelecido pelo Lema 1 definido por Ruiz-Torres, Paletta e M'Hallah (2017).

Neste trabalho, são propostas três versões do algoritmo ILS/D (Algoritmo 4) que tratam o agendamento de tarefas de manutenção. A seguir os três tipos de estratégias utilizadas são explicadas.

4.6.1 Estratégia de Manutenção 1

A Estratégia de manutenção 1 (*Maintenance Strategy 1* - MS1) cria um cenário em que tarefas de manutenção são atribuídas às soluções iniciais. As tarefas de manutenção são removidas da solução selecionada para serem processadas pelo ILS antes da execução do ILS. Assim, o comportamento original do ILS e o tempo de execução disponível do ILS (τ) não foram afetados (alterados). As tarefas de manutenção são adicionadas novamente na nova solução resultante da aplicação do algoritmo ILS. Com este método, é possível determinar se a solução retornada pelo Algoritmo ILS é melhorada quando as tarefas de manutenção são adicionadas. Após a conclusão das tarefas de manutenção, o algoritmo ILS/D continua a ser executado normalmente.

As modificações realizadas no algoritmo ILS/D original para escalonamento de manutenções, de acordo com a estratégia de manutenção 1, são apresentadas no pseudocódigo do Algoritmo 8. Todas as modificações estão na cor vermelha.

Baseado no pseudocódigo do ILS/D - MS1 (Algoritmo 8), as tarefas de manutenção são adicionadas e removidas da seguinte forma:

- As tarefas de manutenção são adicionadas em cada indivíduo da População (P) durante o procedimento construtivo (Linha 4 do Algoritmo ILS/D - MS1)
- As tarefas de manutenção são removidas da solução p passada como parâmetro para o algoritmo ILS (Linha 15 do Algoritmo ILS/D - MS1).
- As tarefas de manutenção são adicionadas na solução P_i retornada pelo algoritmo ILS (Linha 17 do Algoritmo ILS/D - MS1).

Assim sendo, a estratégia de manutenção 1 não considera as tarefas de manutenção na execução do algoritmo ILS (em nenhum dos procedimentos: perturbação, busca local e verificação do melhor vizinho da vizinhança). O tempo computacional necessário para adicionar tarefas de manutenção, de acordo com o Lema 1 definido por Ruiz-Torres, Paletta e M'Hallah (2017), diminui a quantidade de movimento dentro dos procedimentos de perturbação e da busca local. Como resultado, nessa estratégia, foi mantido o tempo

Algoritmo 8: ILS/D - MS1

entrada : s, η, g, τ, t_{min}, t_{max}
saída : P

$W \leftarrow$ gere s vetores de peso usando o método Scheffé;
$P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
para $i = 1$ *até* $i = s$ **faça**
  Adicione tarefas de manutenção em P_i;
fim
para $i = 1$ *até* $i = s$ **faça**
  Associe P_i com w_i;
  $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
fim
$z^* \leftarrow \emptyset$;
Atualize z^* com o melhor valor de cada objetivo;
para $y = 1$ *até* $y = g$ **faça**
  para $i = 1$ *até* $i = s$ **faça**
    Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
    Remove tarefas de manutenção em p;
    $P_i \leftarrow$ ILS(τ, t_{min}, t_{max}, p, w_i, z^*);
    Adicione tarefas de manutenção em P_i;
    Atualize z^* considerando os valores dos objetivos de P_i;
    para *cada vizinho* $w_j \in B_i$ **faça**
      se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
        $P_j \leftarrow P_i$;
      fim
    fim
  fim
fim
Remova as soluções dominadas em P;
retorna P;

de execução do ILS para explorar diferentes soluções, ao invés de adicionar tarefas de manutenção no algoritmo ILS.

4.6.2 Estratégia de Manutenção 2

Como parte da estratégia de manutenção 2 (*Maintenance Strategy 2* - MS2), as tarefas de manutenção são adicionadas (escalonadas) nas soluções iniciais que formam a população e são mantidas na solução selecionada para processamento pelo algoritmo ILS. Antes de executar o procedimento de perturbação do algoritmo ILS, as tarefas de manutenção são removidas da solução corrente. Em seguida, elas são reprogramadas (escalonadas) após o procedimento de perturbação. Antes de gerar a vizinhança, as tarefas de manutenção são removidas da solução corrente durante a execução da busca local.

Ao gerar a vizinhança, tarefas de manutenção são adicionadas a cada solução vizinha. Portanto, a melhor solução encontrada na busca local já inclui tarefas de manutenção.

Devido à natureza computacionalmente intensiva e demorada do escalonamento de tarefas de manutenção, busca-se analisar se é mais apropriado usar todo o tempo de execução (τ) para o algoritmo ILS explorar diferentes soluções (de acordo com a estratégia de manutenção 1), ou se o algoritmo ILS deve usar parte do tempo para adicionar algumas tarefas de manutenção nas soluções.

As modificações realizadas no algoritmo ILS/D original para escalonamento de manutenções, de acordo com a estratégia de manutenção 2, são apresentadas no pseudo-código do Algoritmo 9. Na estratégia de manutenção 2, também foram modificados os algoritmos ILS e de busca local para escalonamento de manutenções. O algoritmo 10 apresenta o pseudo-código modificado do algoritmo ILS original. E o algoritmo 11 apresenta o pseudo-código modificado do algoritmo de busca local original. Todas as modificações estão na cor azul.

Algoritmo 9: ILS/D - MS2

entrada : s, η, g, τ, t_{min}, t_{max}
saída : P

$W \leftarrow$ gere s vetores de peso usando o método Scheffé;
$P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
para $i = 1$ *até* $i = s$ **faça**
    Adicione tarefas de manutenção em P_i;
fim
para $i = 1$ *até* $i = s$ **faça**
    Associe P_i com w_i;
    $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
fim
$z^* \leftarrow \emptyset$;
Atualize z^* com o melhor valor de cada objetivo;
para $y = 1$ *até* $y = g$ **faça**
    para $i = 1$ *até* $i = s$ **faça**
        Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
        $P_i \leftarrow$ ILS(τ, t_{min}, t_{max}, p, w_i, z^*);
        Atualize z^* considerando os valores dos objetivos de P_i;
        para *cada vizinho* $w_j \in B_i$ **faça**
            se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
                $P_j \leftarrow P_i$;
            fim
        fim
    fim
fim
Remova as soluções dominadas em P;
retorna P;

Algoritmo 10: ILS - MS2

entrada : τ, t_{min}, t_{max}, p, w, z^*
saída : $p*$

1 $p \leftarrow$ busca local(p, w, z^*);
2 $p^* \leftarrow p$;
3 $t \leftarrow t_{min}$;
4 **enquanto** *(τ não satisfeito)* **faça**
5 Remove tarefas de manutenção in p;
6 $p' \leftarrow$ perturbação(p, t);
7 Adicione tarefas de manutenção in p';
8 $p' \leftarrow$ busca local(p', w, z^*);
9 **se** $f^{te}(p'|w, z^*) \leq f^{te}(p^*|w, z^*)$ **então**
10 $p^* \leftarrow p'$;
11 $t \leftarrow t_{min}$;
12 **fim**
13 **senão**
14 $t \leftarrow t + 1$;
15 **se** *($t > t_{max}$)* **então**
16 $t \leftarrow t_{min}$;
17 **fim**
18 **fim**
19 **se** $f^{te}(p'|w, z^*) \leq f^{te}(p|w, z^*)$ **então**
20 $p \leftarrow p'$;
21 **fim**
22 **fim**
23 **retorna** p^*;

A abordagem MS2 adiciona e remove tarefas de manutenção da seguinte forma:

- As tarefas de manutenção são adicionadas em cada indivíduo da População (P) durante o procedimento construtivo (Linha 4 do algoritmo ILS/D - MS2).
- As tarefas de manutenção são removidas da solução p antes do procedimento de perturbação ser aplicado (Linha 5 do algoritmo ILS - MS2).
- As tarefas de manutenção são adicionadas na solução p' após o procedimento de perturbação ser computado (Linha 7 do algoritmo ILS - MS2).
- As tarefas de manutenção são removidas da solução p' antes das soluções vizinhas serem criadas (Linha 6 do algoritmo Busca Local - MS2).
- As tarefas de manutenção são adicionadas em cada solução p da população P gerada pela estrutura de vizinhança (Linha 9 do algoritmo Busca Local - MS2).
- As tarefas de manutenção já estão adicionadas na solução p^* retornada pela busca local (Linha 21 do algoritmo Busca Local - MS2).

Algoritmo 11: busca local - MS2

entrada : τ, p', w, z^*
saída : $p*$

1 $p^* \leftarrow p'$;
2 $\Upsilon \leftarrow$ quantidade de estruturas de vizinhanças a serem exploradas;
3 $L \leftarrow$ gere um conjunto com o índice de cada estrutura de vizinhança $\{1, .., \Upsilon\}$;
4 **enquanto** *($L \neq \emptyset$ ou τ não satisfeito)* **faça**
5 $i \leftarrow$ selecione aleatoriamente um índice em L correspondente a uma estrutura de vizinhança;
6 Remove tarefas de manutenção em p';
7 $P \leftarrow$ gere as soluções vizinhas de p' com a estrutura de vizinhança E_i;
8 **para** $p \in P$ **faça**
9 Adicione tarefas de manutenção em p;
10 **se** $f^{te}(p|w, z^*) < f^{te}(p'|w, z^*)$ **então**
11 $p' \leftarrow p$;
12 **fim**
13 **fim**
14 **se** $f^{te}(p'|w, z^*) < f^{te}(p^*|w, z^*)$ **então**
15 $p^* \leftarrow p'$ e $L \leftarrow \{1, .., \Upsilon\}$;
16 **fim**
17 **senão**
18 $L \leftarrow L/\{i\}$;
19 **fim**
20 **fim**
21 **retorna** p^*;

- As tarefas de manutenção já estão adicionadas na solução P_i retornada pelo algoritmo ILS (Linha 15 do algoritmo ILS/D - MS2).

Consequentemente, a estratégia de manutenção 2 não leva em consideração as tarefas de manutenção durante a execução dos procedimentos de perturbação e busca local do algoritmo ILS. No entanto, dentro do algoritmo ILS, as tarefas de manutenção são consideradas ao selecionar o melhor vizinho de toda a vizinhança gerada pela busca local. Portanto, busca-se verificar se a inclusão de tarefas de manutenção nas soluções da vizinhança geradas pela busca local pode melhorar a qualidade das soluções vizinhas. Além disso, busca-se evitar mudanças nas posições das tarefas de manutenção durante a execução de procedimentos de busca local e perturbação. Isso pode incluir, por exemplo, garantir que a tarefa de manutenção não seja adicionada antes da primeira tarefa atribuída à máquina ou após a última tarefa atribuída à máquina, isto é, quando não há tarefas subsequentes. Também é importante garantir que as tarefas de manutenção não sejam adicionadas imediatamente uma após a outra e não violem o Lema 1 conforme definido por Ruiz-Torres, Paletta e M'Hallah (2017).

4.6.3 Estratégia de Manutenção 3

A estratégia de manutenção 3 (*Maintenance Strategy 3* - MS3) cria um cenário em que as tarefas de manutenção são adicionadas (escalonadas) nas soluções iniciais que formam a população e são mantidas durante a execução do ILS/D. Essa abordagem também permite que o algoritmo ILS altere as posições das tarefas de manutenção por meio dos procedimentos de perturbação e busca local. Esta estratégia foi proposta para verificar se a alteração das tarefas de manutenção dentro das soluções do ILS permite encontrar soluções melhores do que aquelas encontradas por meio dos procedimentos de perturbação e busca local sem considerar as tarefas de manutenção. Isso permite que as tarefas de manutenção sejam tratadas livremente e da mesma forma que outras tarefas da máquina.

As modificações realizadas no algoritmo ILS/D original para escalonamento de manutenções, de acordo com a estratégia de manutenção 3, são apresentadas no pseudo-código do Algoritmo 12. Todas as modificações estão na cor laranja.

Assim, a estratégia de manutenção 3 adiciona, mantém e altera as tarefas de manutenção da seguinte forma:

- As tarefas de manutenção são adicionadas em cada indivíduo da População (P) durante o procedimento construtivo (Linha 4 do algoritmo ILS/D - MS3).
- As tarefas de manutenção são mantidas e enviadas na solução p passada como parâmetro para o algoritmo ILS (Linha 15 do Algoritmo ILS/D - MS3).
- As tarefas de manutenção são mantidas na solução p e alteradas na solução p' obtida após a execução do procedimento de perturbação.
  - No procedimento de perturbação, uma tarefa de manutenção pode ser adicionada ou removida de acordo com a proporção de manutenções já realizadas na máquina. Para determinar a taxa de manutenção de uma máquina, divide-se a quantidade de manutenções pelo número de tarefas que ela executa. Como próximo passo, seleciona-se aleatoriamente um valor entre 0 e 100. Caso esse valor exceda a taxa de manutenção da máquina, uma tarefa de manutenção será adicionada em uma das posições da máquina que ainda não contém uma tarefa de manutenção. Por outro lado, se o valor for menor que a taxa de manutenção da máquina, uma posição das manutenções existentes é removida da máquina (a escolha entre essas posições é aleatória). Como resultado, procura-se alcançar um equilíbrio entre o número de tarefas e de manutenção.
- As tarefas de manutenção são mantidas na solução p' e alteradas nas soluções vizinhas geradas pela estrutura de vizinhança.

Algoritmo 12: ILS/D - MS3

entrada : s, η, g, τ, t_{min}, t_{max}
saída : P

$W \leftarrow$ gere s vetores de peso usando o método Scheffé;
$P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
para $i = 1$ *até* $i = s$ **faça**
  Adicione tarefas de manutenção em P_i;
fim
para $i = 1$ *até* $i = s$ **faça**
  Associe P_i com w_i;
  $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
fim
$z^* \leftarrow \emptyset$;
Atualize z^* com o melhor valor de cada objetivo;
para $y = 1$ *até* $y = g$ **faça**
  para $i = 1$ *até* $i = s$ **faça**
    Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
    $P_i \leftarrow$ ILS(τ, t_{min}, t_{max}, p, w_i, z^*);
    Atualize z^* considerando os valores dos objetivos de P_i;
    para *cada vizinho* $w_j \in B_i$ **faça**
      se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
        $P_j \leftarrow P_i$;
      fim
    fim
  fim
fim
Remova as soluções dominadas em P;
retorna P;

  - Na busca local, as tarefas de manutenção são trocadas com as outras tarefas da máquina, formando novos vizinhos de acordo com cada estrutura de vizinhança.

- As tarefas de manutenção são mantidas na solução p' e nas soluções vizinhas que formam a população P para comparar qual é a melhor solução.

Portanto, a estratégia de manutenção 3 leva em consideração as tarefas de manutenção ao executar o algoritmo ILS. As posições das tarefas de manutenção são alteradas durante a execução dos procedimentos de perturbação e busca local, bem como durante qualquer outra operação.

Com essa estratégia, busca-se verificar se soluções mais diversificadas podem ser encontradas ao processar as tarefas de manutenção como outras tarefas do sequenciamento. Portanto, as seguintes regras utilizadas em MS1 e MS2 não foram implantadas nesta estratégia: uma tarefa de manutenção pode ser adicionada logo após outra tarefa de

manutenção, e o Lema 1 definido por Ruiz-Torres, Paletta e M'Hallah (2017) foi violado (infringido).

4.7 Conclusão

Problemas de programação da produção com efeitos de deterioração têm sido objeto de extensiva pesquisa ao longo dos últimos anos. Em geral, os estudos têm se concentrado em desenvolver algoritmos mais rápidos e precisos para resolver o problema de forma mono-objetivo. No entanto, o foco deste capítulo foi direcionado à proposição de um algoritmo multiobjetivo para abordar essa complexa questão.

Considerando a natureza NP-difícil do problema, a resolução de instâncias de grande porte em busca da otimalidade torna-se impraticável. Para obter soluções quase ótimas, foi desenvolvido um algoritmo heurístico específico, denominado ILS/D. É relevante destacar que a abordagem de decomposição e agregação empregada no MOEA/D desempenha um papel crucial no controle da diversificação da aproximação da fronteira Pareto, e o ILS/D mantém essa característica intacta. Além disso, o algoritmo híbrido ILS/D foi adaptado para incorporar a tarefa de manutenção ao problema multiobjetivo, resultando no desenvolvimento de três estratégias distintas para o sequenciamento das tarefas de manutenção.

O trabalho realizado neste capítulo estabelece uma base sólida para a solução eficaz do problema de sequenciamento de produção com deterioração, considerando dois objetivos e a integração de tarefas de manutenção. As estratégias de otimização desenvolvidas proporcionam uma abordagem inovadora e promissora para lidar com os desafios inerentes a essa classe de problemas. No capítulo seguinte, são apresentados os resultados dos experimentos computacionais, permitindo uma avaliação abrangente do desempenho dos algoritmos propostos.

5 Experimentos

5.1 Introdução

Neste capítulo, é conduzida uma análise rigorosa e detalhada dos experimentos computacionais realizados para avaliar o desempenho dos algoritmos desenvolvidos neste trabalho. Os experimentos são fundamentais para validar e demonstrar a eficácia das estratégias propostas para o sequenciamento de produção com deterioração, bem como para a integração das operações de manutenção, ou seja, das tarefas de manutenção.

Ao longo deste capítulo, é fornecida uma visão abrangente das instâncias do problema utilizadas, analisados os parâmetros críticos do algoritmo ILS/D e apresentados os resultados obtidos a partir das execuções dos algoritmos em diferentes cenários. A comparação desses resultados com outros algoritmos de referência, como MOEA/D, NSGA-II e PILS, é uma parte fundamental da nossa análise. Além disso, são discutidos os efeitos das estratégias de sequenciamento de manutenção propostas.

5.2 Experimentos Computacionais I

Nesta seção são descritos os experimentos realizados a fim de estudar o comportamento do algoritmo ILS/D proposto. A principal questão de pesquisa que pretende-se explorar é a seguinte: a abordagem que emprega decomposição e agregação e usa um ILS mono-objetivo para melhorar cada subproblema é a mais eficiente para o problema multiobjetivo proposto? Para responder a essa questão, o algoritmo ILS/D é comparado com os algoritmos MOEA/D e NSGA-II, duas meta-heurísticas baseadas em população amplamente utilizados para resolver problemas multiobjetivo. O ILS/D é também comparado com o algoritmo PILS, o qual segue a estrutura comum do ILS, mas depende de uma busca local baseada em dominância Pareto. Assim, a resposta a esta questão pode contribuir com uma importante implicação para meta-heurísticas baseadas em busca local relacionadas aos problemas de otimização multiobjetivo.

Os algoritmos heurísticos desenvolvidos foram codificados em C ++ e executados em cinco computadores independentes e idênticos (software e hardware), com configuração: Intel® Core^{TM} i7-870, CPU 2,93 GHz, 8GB de RAM, sistema operacional Ubuntu 16.04. 5 LTS.

O desempenho de cada algoritmo é medido pelo cálculo da métrica de hipervolume (HV) proposta por Zitzler e Thiele (1999). No caso de duas funções objetivo, a métrica de hipervolume fornece um valor, que é definido como a área do polígono delimitada

pela fronteira Pareto aproximada de todas as soluções candidatas não dominadas viáveis e um determinado ponto de referência. Assim, quanto maior o valor de HV, melhor a qualidade das soluções (VARGAS et al., 2016). O ponto de referência usado para calcular o HV é composto pelos extremos das soluções encontradas para cada um dos objetivos. Em outras palavras, esse ponto é um vetor composto pelo valor mínimo (quando se trata de um objetivo a ser minimizado) ou pelo valor máximo (quando se trata de um objetivo a ser maximizado) de cada objetivo, considerando as soluções extremas conhecidas até o momento. Todas as soluções encontradas pelos algoritmos desenvolvidos foram utilizadas para identificar os valores mínimo e máximo de cada objetivo nas fronteiras Pareto aproximadas. Os valores do indicador HV foram normalizados entre 0 e 1.

Nas subseções seguintes são apresentadas: (i) a geração das instâncias do problema; (ii) as configurações dos parâmetros do algoritmo ILS/D; (iii) a análise dos resultados obtidos pelos algoritmos ILS/D e MOEA/D; (iv) a comparação dos resultados obtidos pelo algoritmo NSGA-II; (v) a análise dos resultados obtido pelo algoritmo PILS; e (vi) a validação dos algoritmos a partir do conhecimento das soluções ótimas obtidas pela técnica de enumeração.

5.2.1 Instâncias do Problema

Os algoritmos foram testados em instâncias propostas por Ruiz-Torres, Paletta e Perez-Roman (2015), as quais estão disponíveis em ⟨https://ruiz-torres.uprrp.edu/nt/⟩. O tamanho de uma instância é definido pelo número de tarefas (n) e número de máquinas (m). As instâncias de Ruiz-Torres, Paletta e Perez-Roman (2015) usam combinações dos seguintes valores de n e m: $n \in \{100, 200\}$ e $m \in \{5, 10, 20\}$.

Nessas instâncias, os tempos de processamento p_{jk} são números aleatórios, com distribuição uniforme, no intervalo [1; 100]. Os efeitos de deterioração d_{jk} também são números aleatórios, com distribuição uniforme em dois intervalos: [1%; 5%] e [5%; 10%].

Para atribuir as datas de vencimento de cada tarefa, Ruiz-Torres, Paletta e Perez-Roman (2015) consideraram o tempo médio de processamento ($\bar{p}$), o número de recursos e a taxa de congestionamento, denominada *congestion ratio* - *CR*. A CR é usada para estabelecer o aperto total da data de vencimento; uma CR mais alta resultaria em um número maior de tarefas atrasadas. Tem-se que: $D_{\max} = \frac{\bar{p} \times n}{m \times CR}$. Assim, a data de vencimento de uma tarefa é determinada por: ${p_j}^{\min} + U(0, D_{\max})$, onde ${p_j}^{\min} = \min_{k \in M}\{p_{jk}\}$. Este método de atribuição das datas de vencimento garante que cada tarefa pode estar dentro do prazo (*on-time*) se pelo menos estiver alocada em uma das máquinas no início do sequenciamento, portanto, não há tarefas atrasadas inerentes. Os valores de CR das datas de entrega e_j para cada tarefa j são dados por: $CR \in \{2; 3, 5; 5\}$ (RUIZ-TORRES; PALETTA; PEREZ-ROMAN, 2015).

Utilizando os parâmetros n, m, intervalo de tempo de processamento, intervalo de efeito de deterioração e taxa de congestionamento de datas de vencimento, foram obtidas 36 combinações. Para cada combinação, 25 instâncias foram geradas, resultando em um total de 900 instâncias.

5.2.2 Análise dos Parâmetros do Algoritmo ILS/D

Huang, Li e Yao (2019) definem três métodos para realizar o ajuste de parâmetros: métodos simples de gerar-avaliar; métodos interativos de gerar-avaliar; e métodos de geração e avaliação de alto nível. Neste trabalho, o método simples de gerar-avaliar é usado. Este método consiste em gerar um conjunto de configurações candidatas e avaliar cada uma dessas configurações para encontrar a melhor. Entre os métodos simples de gerar-avaliar, existe a abordagem de força bruta.

De acordo com Birattari et al. (2002), a abordagem de força bruta é o método mais fácil de ajustar parâmetros, mas este método pode ter algumas desvantagens. Por exemplo, baixa eficiência no uso do poder computacional e dificuldade em definir quantas execuções de cada configuração em cada instância devem ser realizadas.

Apesar das desvantagens mencionadas, optou-se por adotar a abordagem de força bruta neste trabalho, devido ao conhecimento prévio sobre um conjunto potencial de parâmetros promissores. Adicionalmente, a simplicidade de implementação e a capacidade de realizar execuções simultâneas com essa abordagem permitiram a aplicação de diversos testes estatísticos para uma análise aprofundada dos resultados obtidos. Nesse contexto, utilizar abordagens que explorem uma ampla gama de combinações de parâmetros, como o pacote irace (LÓPEZ-IBÁÑEZ et al., 2016), se tornaria inviável devido ao tempo computacional exigido para a execução de cada combinação de parâmetros no escopo deste trabalho.

Quando um algoritmo é calibrado nas mesmas instâncias que serão usadas posteriormente para resultados computacionais e comparações, corre-se o risco de ter resultados enviesados ou super ajustados (JACOB, 2014). A fim de remediar este problema, foi gerado um *benchmark* de calibração de 36 instâncias aleatórias com as combinações possíveis entre n, m, d_{jk}, p_{jk} e CR, respectivamente ($2 \times 3 \times 2 \times 1 \times 3 = 36$).

Para calibração, os algoritmos foram executados 5 vezes para cada instância. A calibração foi realizada usando um fator de cada vez. Para cada parâmetro, um conjunto de valores foi testado fixando os outros.

Para obter o melhor desempenho do algoritmo ILS/D, os seus parâmetros de entrada foram ajustados. O ILS/D tem seis parâmetros de entrada a serem definidos:

- s: define o número de subproblemas (e, portanto, o número de vetores de peso)

Tabela 4 – Conjunto de valores dos parâmetros para calibração do algoritmo ILS/D

Parâmetro	Valores	Quantidade
g	$\{30; 40; 50\}$	3
s	$\{30; 40; 50\}$	3
η	$\{6; 8; 10\}$	3
t_{min}	$\{2\}$	1
t_{max}	$\{0,7 \times m; 0,9 \times m\}$	2
p_{fixed}	$\{1; 0\}$	2
Total		108

gerados pela decomposição para o algoritmo ILS/D.

- η: define o número de vizinhos para cada subproblema gerado no algoritmo ILS/D.
- g: define o número máximo de gerações para o algoritmo ILS/D.
- τ: controla o tempo máximo de execução para o algoritmo ILS.
- t_{min}: define o tamanho mínimo de perturbação para o algoritmo ILS.
- t_{max}: define o tamanho máximo de perturbação para o algoritmo ILS.
- p_{fixed}: define se o tamanho de perturbação do algoritmo ILS deve permanecer fixo ou variar.

É usado um critério de parada natural (τ), o qual foi definido pelo tempo de CPU decorrido, seguindo a expressão $0,08 \times \frac{n}{m}$ segundos. Nesta expressão, o tempo de CPU é influenciado pelo número de tarefas n e pelo número de máquinas m. Para n fixado, o tempo de execução aumenta à medida que o valor de m decresce. O parâmetro t_{min} foi fixado em 2, porque é o número mínimo de máquinas a serem carregadas para realizar o procedimento de perturbação. O parâmetro p_{fixed} é definido igual a 1 se o nível de perturbação permanece fixo e 0, caso contrário. Os demais parâmetros foram calibrados por meio de experimentos computacionais. Os valores testados para os parâmetros são especificados na Tabela 4. Observe que há um total de 108 combinações de parâmetros para avaliar. Como foram utilizadas 36 instâncias, 5 execuções de cada e 108 configurações, um total de 19.440 execuções foram realizadas.

Para avaliar a significância estatística das diferenças observadas entre os resultados dos algoritmos, foram verificadas as três premissas fundamentais para a aplicação de testes paramétricos: normalidade dos dados, homogeneidade de variâncias e independência dos resíduos. No entanto, o teste *Shapiro-Wilk* (SHAPIRO; WILK, 1965) revelou que os dados não seguem uma distribuição normal, impossibilitando o uso da Análise de Variância (ANOVA). Diante dessa constatação, optou-se pela aplicação de métodos não paramétricos

para análise. Neste sentido, o teste *Kruskal-Wallis* (KRUSKAL; WALLIS, 1952) foi adotado. Ao aplicar este teste obteve-se um *p-value* inferior a 0,05, indicando a existência de diferenças estatisticamente significativas entre as configurações dos algoritmos, com um nível de confiança de 95%. Tal resultado sugere que, no mínimo, dois grupos apresentam desempenhos distintos.

Como abordado, para identificar qual algoritmo obteve o maior HV médio e verificar a existência de diferenças estatísticas entre os algoritmos, recorreu-se ao teste não paramétrico *Kruskal-Wallis*. Esse teste foi realizado acompanhado de análises *post hoc*[1], juntamente com a geração de um *box-plot*, que oferece uma visão concisa da distribuição dos dados, realçando a mediana, os quartis e os *outliers*. A integração do *box-plot* com os resultados do teste *post hoc* permite a inclusão de letras acima dos boxes, indicativas de diferenças significativas entre os grupos avaliados, em que: letras idênticas sugerem ausência de diferenças estatísticas significativas entre os grupos, enquanto letras distintas apontam para a existência de tais diferenças. Neste trabalho, pode-se interpretar que grupos (algoritmos e/ou parametrização) marcados com a letra "a" são considerados de desempenho superior e estatisticamente distintos dos grupos marcados com "b", e assim sucessivamente, indicando uma ordem de eficácia baseada em desempenho estatisticamente validado.

A Figura 6 mostra o gráfico médio resultante do teste *Box-plot of Kruskal-Wallis groups*, com um nível de confiança de 95% para as 108 configurações testadas do algoritmo ILS/D. É possível ver na Figura 6 que existe uma diferença significativa entre as configurações, pois existem muitas classificações de grupos de intervalos diferentes (representados por letras distintas). Por exemplo, existem diferenças significativas entre as configurações 108 e 01 (grupo a e jk, respectivamente); 52 e 55 (grupo abc e $GHIJK$, respectivamente). Percebe-se também que não existe uma única configuração que seja estatisticamente melhor do que todas as outras, uma vez que não existe apenas uma configuração classificada como o grupo a.

Dentre as configurações pertencentes ao grupo a e influenciadas pelos parâmetros s e g, que impactam diretamente no tempo, a configuração mais promissora identificada é a de número 99. Esta configuração apresenta os valores dos parâmetros $s = 40$ e $g = 50$, e está associada ao grupo $abcde$, com uma média de $0,8262$. A configuração 108, com uma média de $0,8363$, é considerada a melhor em termos de desempenho médio. É importante destacar que a diferença entre as médias das configurações 108 e 99 é relativamente pequena, e a configuração 99 possui um valor menor para o parâmetro s, que influencia no tempo de execução. Além disso, com um nível de confiança de 95%, as médias dessas

[1] Testes *post hoc* são análises subsequentes realizadas após um teste estatístico global (como o *Kruskal-Wallis*) para indicar diferenças significativas, com o objetivo de identificar entre quais grupos específicos essas diferenças ocorrem. Neste trabalho, foi utilizado a função *kruskal* da biblioteca agricolae (Disponível em: https://rdrr.io/cran/agricolae/man/kruskal.html).

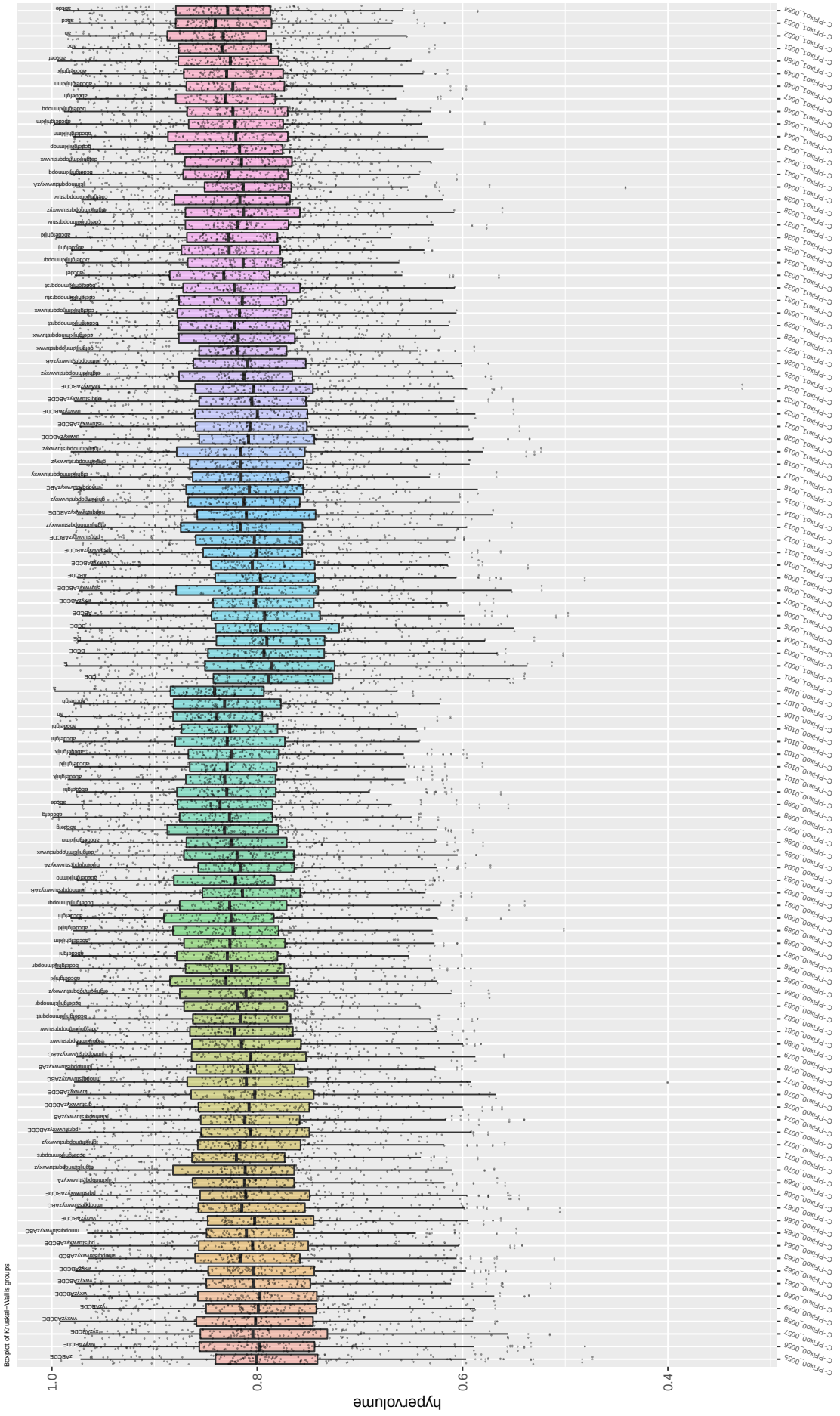


Figura 6 – Gráfico de Médias e intervalos do teste *Box-plot of Kruskal-Wallis groups* para a calibração dos parâmetros do algoritmo ILS/D.

configurações são estatisticamente equivalentes. Portanto, para os experimentos finais do algoritmo ILS/D, foram usados os parâmetros da configuração 99: $\tau = 0,08 \times \frac{n}{m}$ segundos, $g = 50$; $s = 40$; $\eta = 8$; $t_{min} = 2$, $t_{max} = 0,7 \times m$, $p_{fixed} = 0$ (isso significa que o nível de perturbação deve variar entre t_{min} e t_{max}).

5.2.3 Comparação entre os Resultados dos Algoritmos ILS/D e MOEA/D

Nesta subseção, os algoritmos ILS/D e MOEA/D são avaliados em 900 instâncias da literatura (RUIZ-TORRES; PALETTA; M'HALLAH, 2017). Cada grupo contém 25 instâncias, e cada uma delas foi submetida a cinco execuções, totalizando 125 repetições. Apenas as médias dos resultados obtidos nessas execuções foram considerados.

Para realizar os testes com o algoritmo MOEA/D, foram criados operadores de cruzamento e mutação. Em relação ao cruzamento, ele foi parcialmente baseado no operador de cruzamento para sequenciamento de uma única máquina apresentado em Sadegheih (2006). Em relação ao operador de mutação, os seis exemplos de movimentos das estruturas de vizinhanças, apresentados na Seção 4.5.2.2, foram selecionados aleatoriamente e aplicados na prole (*offspring*). O Apêndice B ilustra os processos de cruzamento e mutação implementados neste trabalho.

Os parâmetros utilizados nos testes com o algoritmo MOEA/D, inicialmente, foram os mesmos parâmetros calibrados para o algoritmo ILS/D, ou seja, $g = 50$; $s = 40$ e $\eta = 8$. No entanto, notou-se que o algoritmo MOEA/D com esses parâmetros apresentou resultados com desempenho inferior ao algoritmo ILS/D. Portanto, testes do MOEA/D com o maior número de gerações, ou seja, $g = 500$, $g = 5000$, $g = 50000$ e $g = 500000$ foram executados. A variação do valor do parâmetro g foi conduzida com o objetivo de estimar o número de gerações necessárias para que o algoritmo MOEA/D alcance um desempenho próximo ao obtido pelo algoritmo ILS/D, expresso em termos dos valores de HV.

A Tabela 5 apresenta os resultados médios obtidos para a métrica HV. Os HV médios são agrupados por tamanho de instância ($n \times m \times d_{jk} \times CR$), onde para cada combinação encontram-se 25 instâncias. Os valores em negrito representam os melhores resultados.

Na Tabela 5, o parâmetro g referente ao número de gerações é fixo para o algoritmo ILS/D ($g = 50$) e variável para o algoritmo MOEA/D ($g = 50$, $g = 500$, $g = 5000$, $g = 50000$, $g = 500000$). Pode-se ver que o ILS/D ($g = 50$) apresenta os melhores resultados para todos os problemas de teste, quando comparado com o MOEA/D, com os parâmetros $g = 50$, $g = 500$ e $g = 5.000$. Por outro lado, quando é comparado o ILS/D e o MOEA/D ($g = 50000$), o algoritmo MOEA/D apresenta melhores resultados para os problemas de teste 100×20 e 200×20.

Tabela 5 – Resultados do indicador médio HV para os algoritmos ILS/D e MOEA/D.

n	m	d_{jk}	CR	ILS/D $g = 50$	MOEA/D $g = 50$	MOEA/D $g = 500$	MOEA/D $g = 5000$	MOEA/D $g = 50000$	MOEA/D $g = 500000$
100	20	[1%, 5%]	2	0,8432	0,6143	0,7627	0,8611	**0,9047**	0,9011
			3,5	0,7569	0,4035	0,6292	0,7719	0,8657	**0,8927**
			5	0,7235	0,3706	0,6003	0,7610	0,8612	**0,8939**
		[5%, 10%]	2	0,8401	0,5469	0,7058	0,8169	0,8675	**0,8771**
			3,5	0,8038	0,4365	0,6482	0,7817	0,8854	**0,9180**
			5	0,7497	0,3686	0,6038	0,7531	0,8614	**0,9117**
	10	[1%, 5%]	2	0,9117	0,4308	0,6812	0,8214	0,8897	**0,9240**
			3,5	0,8566	0,3135	0,6189	0,7681	0,8715	**0,9115**
			5	0,8025	0,3113	0,5943	0,7950	0,8749	**0,9175**
		[5%, 10%]	2	0,9254	0,3666	0,6694	0,8016	0,8964	**0,9386**
			3,5	0,8693	0,3443	0,6233	0,7813	0,8544	**0,9075**
			5	0,8513	0,3866	0,6148	0,7727	0,8433	**0,9018**
	5	[1%, 5%]	2	**0,9160**	0,3061	0,5801	0,7421	0,8366	0,8774
			3,5	**0,8770**	0,3479	0,5749	0,7188	0,8089	0,8678
			5	**0,8687**	0,3701	0,5596	0,7172	0,7977	0,8582
		[5%, 10%]	2	**0,8908**	0,2624	0,5291	0,7189	0,7963	0,8509
			3,5	**0,8877**	0,3467	0,5553	0,7278	0,8009	0,8564
			5	**0,8822**	0,3625	0,5636	0,7327	0,8047	0,8572
200	20	[1%, 5%]	2	0,9052	0,5282	0,7424	0,8395	0,9113	**0,9587**
			3,5	0,8569	0,3427	0,6058	0,7510	0,8786	**0,9417**
			5	0,8401	0,3051	0,5739	0,7410	0,8705	**0,9498**
		[5%, 10%]	2	0,9261	0,4190	0,6983	0,8295	0,9084	**0,9616**
			3,5	0,8993	0,3211	0,5913	0,7837	0,8712	**0,9478**
			5	0,8343	0,2976	0,5380	0,7702	0,8693	**0,9288**
	10	[1%, 5%]	2	0,9457	0,3421	0,6211	0,7815	0,8891	**0,9519**
			3,5	0,8601	0,2954	0,5085	0,7315	0,8281	**0,8806**
			5	0,8315	0,3510	0,5220	0,7428	0,8421	**0,8878**
		[5%, 10%]	2	**0,9019**	0,1694	0,3782	0,7045	0,8159	0,8616
			3,5	**0,8517**	0,2520	0,3989	0,6579	0,7861	0,8376
			5	**0,8531**	0,2895	0,4453	0,6448	0,7804	0,8383
	5	[1%, 5%]	2	**0,8843**	0,2195	0,4013	0,6922	0,7887	0,8615
			3,5	**0,8717**	0,2830	0,4409	0,6855	0,7783	0,8459
			5	**0,8718**	0,2974	0,4621	0,6894	0,7776	0,8456
		[5%, 10%]	2	**0,9161**	0,0712	0,3264	0,7060	0,8157	0,8831
			3,5	**0,9209**	0,0845	0,3497	0,7213	0,8278	0,8933
			5	**0,9264**	0,0892	0,3788	0,7380	0,8434	0,9002

Além disso, a Tabela 5 mostra que quando compara-se o ILS/D ($g = 50$) e o MOEA/D ($g = 500000$), o algoritmo MOEA/D apresenta melhores resultados para os problemas de teste 100×20, 100×10 e 200×20. Isso ocorre porque quando o valor n é fixo e o valor m é incrementado, o algoritmo ILS/D terá menos tempo para executar. Assim, o ILS/D teve menos tempo para convergir nesses problemas de teste. Para os problemas de teste 200×10 os dois algoritmos são similares. E para os problemas de teste 100×5 e 200×5, o algoritmo ILS/D obtém melhores resultados.

No entanto, entende-se que para o valor de n fixo, quando o valor de m é diminuído, mais difícil é para resolver o problema, por exemplo, 100×5 e 200×5. Assim, o algoritmo ILS/D tem um bom desempenho para resolver os problemas mais difíceis. Os resultados sugerem que o algoritmo ILS/D tem melhor desempenho do que o algoritmo MOEA/D porque o algoritmo MOEA/D precisa de mais gerações do que o ILS/D para obter um bom desempenho.

Para validar os resultados obtidos pelos algoritmos e verificar se as diferenças observadas são estatisticamente significativas, aplicou-se o teste não paramétrico *Kruskal-Wallis* para os grupos de tamanho de instância ($n \times m$), em que para cada combinação encontram-se 150 instâncias, ou seja, 150 amostras que foram executadas 5 vezes cada uma, totalizando 750 repetições, e apenas o resultado médio foi considerado. Verificou-se que os dados não seguem uma distribuição normal usando o teste *Shapiro-Wilk*. Como já mencionado, o teste *Kruskal-Wallis* é o método não paramétrico alternativo à ANOVA para um fator. O *p-value* resultante obtido no teste *Kruskal-Wallis* foi menor que 0,05 para todos os problemas do teste. Os resultados sugerem que há uma diferença estatística entre os resultados do indicador HV para os algoritmos comparados com nível de confiança de 95% (limiar = 0,05).

Para identificar diferenças significativas entre os grupos dos algoritmos, e indicar quais pares específicos de grupos diferem entre si, foi realizado o teste *Box-plot of Kruskal-Wallis groups* com nível de confiança de 95%, juntamente com os resultados do teste *post hoc*. A Figura 7 mostra esta análise a partir da comparação dos algoritmos ILS/D e MOEA/D para os problemas teste 100×20 (Figura 7-a), 100×10 (Figura 7-b), 100×5 (Figura 7-c), 200×20 (Figura 7-d), 200×10 (Figura 7-e) e 200×5 (Figura 7-f).

Na Figura 7 (a), (b) e (d), visto que o intervalo para o algoritmo MOEA/D ($g = 500000$) é classificado como o grupo a e o intervalo para o algoritmo ILS/D ($g = 50$) é classificado como grupo b ou c, a média do algoritmo MOEA/D é significativamente diferente. Ou seja, o MOEA/D apresentou desempenho superior ao ILS/D.

Na Figura 7 (e), para os problemas teste 200×10, pode-se ver que os algoritmos não apresentam diferenças significativas quando comparados entre si, pois todas as configurações fazem parte do grupo a.

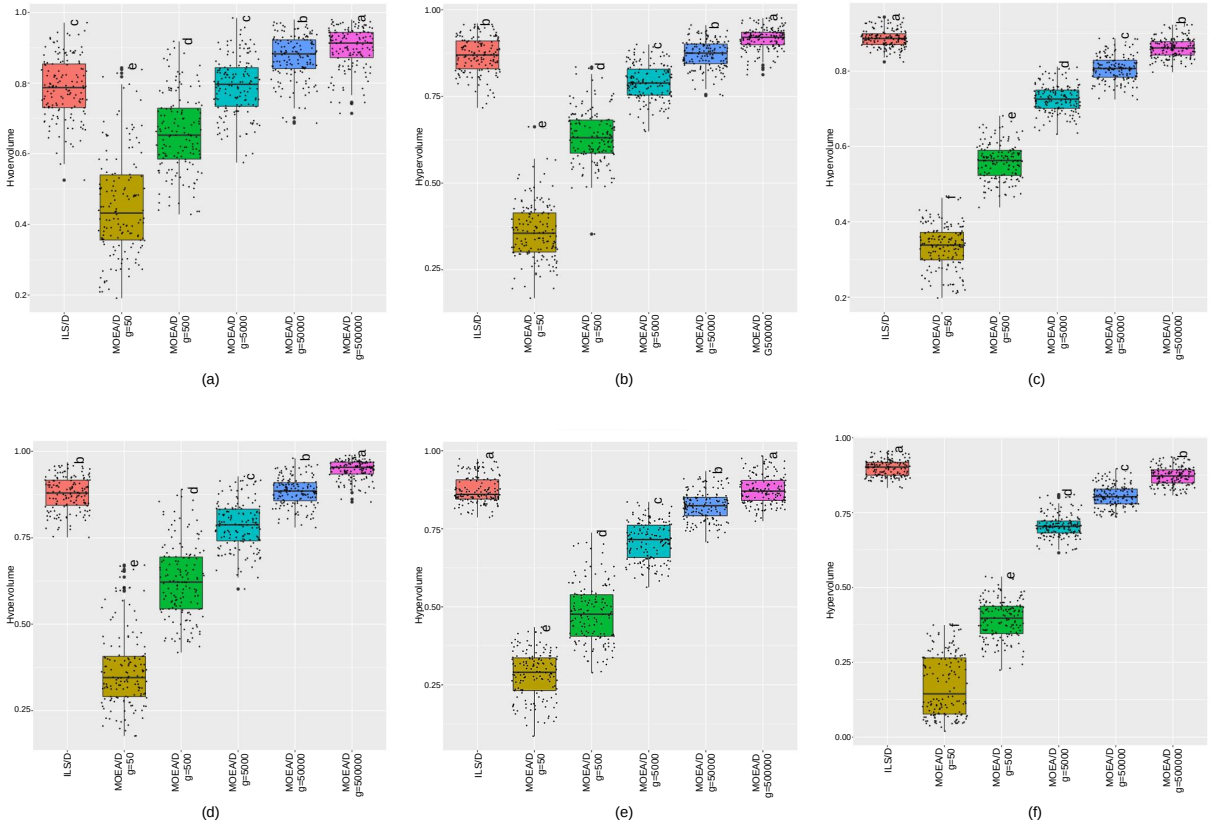


Figura 7 – Resultados do teste *Box-plot of Kruskal-Wallis groups* dos algoritmos ILS/D e MOEA/D para o indicador HV, dos problemas teste $n \times m$.

Na Figura 7 (c) e (f), percebe-se que o intervalo para o algoritmo ILS/D ($g = 50$) é classificado como o grupo a para os problemas teste 100×5 e 200×5, e o intervalo para o MOEA/D ($g = 50$, $g = 500$, $g = 5000$, $g = 50000$ e $g = 500000$) são classificados como os grupos f, e, d, c e b, respectivamente. Ou seja, o algoritmo ILS/D apresenta um desempenho melhor que os algoritmos MOEA/D quando comparados entre si.

Em geral, os resultados do teste *Box-plot of Kruskal-Wallis groups* sugerem que o ILS/D ($g = 50$) é estatisticamente melhor que o MOEA/D ($g = 50$, $g = 500$, $g = 5000$ e $g = 50000$), em uma comparação par-a-par. Entretanto, quando o ILS/D ($g = 50$) é comparado com o MOEA/D ($g = 500000$), o ILS/D apresenta resultados superiores apenas para os problemas teste 100×5 e 200×5, que são considerados os problemas mais difíceis de resolver. Este resultado também sugere a eficiência do algoritmo proposto nas condições definidas do experimento.

Foi realizada uma análise gráfica da convergência dos algoritmos. Seis instâncias ($n \times m$) foram selecionadas aleatoriamente para ilustrar os resultados. A Figura 8 mostra os resultados da melhor execução entre as cinco execuções das instâncias selecionadas. A Figura 8 sugere que o algoritmo MOEA/D ($g = 500000$) tem uma boa convergência para a fronteira Pareto, considerando o ambiente experimental para os problemas de teste 100×20 (Figura 8-a), 100×10 (Figura 8-b) e 200×20 (Figura 8-d).

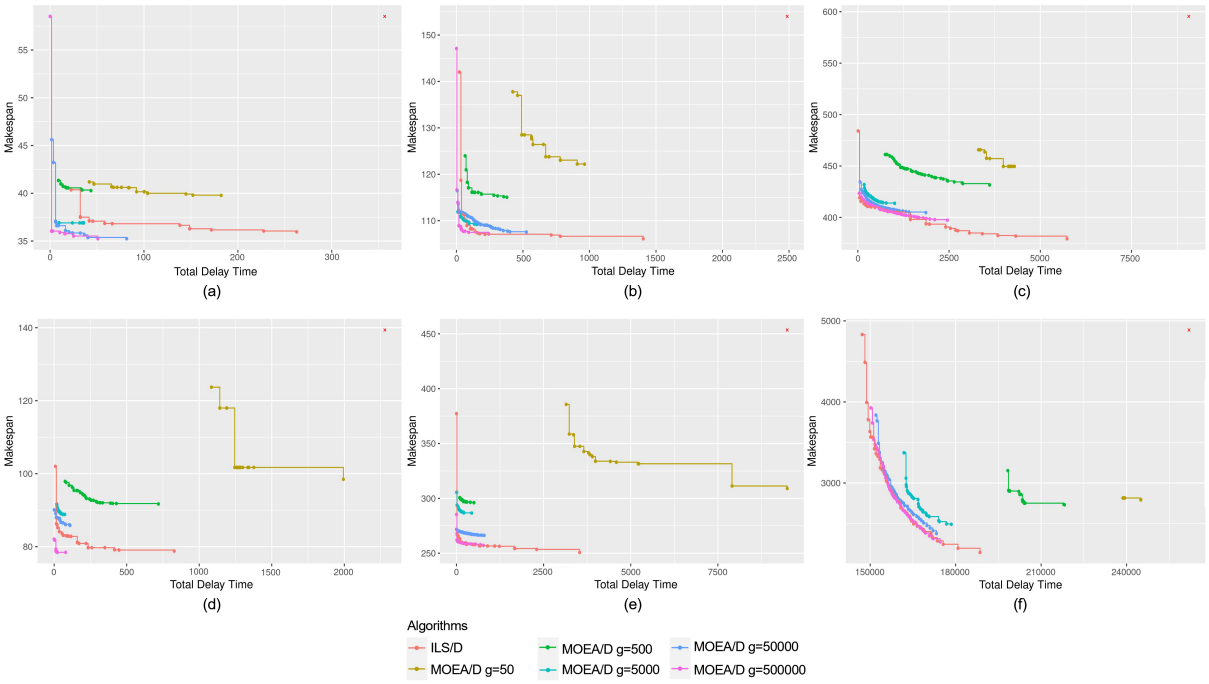


Figura 8 – Fronteiras Pareto aproximadas obtidas com os algoritmos ILS/D e MOEA/D para os problemas teste $n \times m$.

Por outro lado, a Figura 8 sugere que o algoritmo ILS/D ($g = 50$) tem uma boa convergência para a fronteira Pareto, considerando o ambiente experimental para os problemas de teste 100×5 (Figura 8-c), 200×10 (Figura 8-e) e 200×5 (Figura 8-f).

Além disso, a Figura 8 sugere que as fronteiras Pareto aproximadas obtidas pelo ILS/D, na maioria dos casos, possuem uma extensão melhor. Embora o algoritmo MOEA/D obtenha um valor melhor para o indicador de hipervolume para algumas instâncias, pode-se verificar que há uma pobre extensão da fronteira Pareto aproximada. Entende-se que um diferencial do algoritmo MOEA/D é a setorização da busca na fronteira Pareto aproximada. Esta mesma característica é observada no algoritmo ILS/D. No entanto, pode-se observar que a fronteira Pareto aproximada do ILS/D possui extensão maior quando comparado com o MOEA/D, provavelmente devido ao benefício de empregar uma busca local baseada em vizinhança e buscar a diversificação de soluções (método de perturbação) para o paralelismo implícito baseado na população e decomposição como o algoritmo MOEA/D.

Um segundo indicador também é usado para avaliar a qualidade das fronteiras Pareto aproximadas obtidas pelos algoritmos. Este indicador é o cálculo do valor da distância Euclidiana entre os dois pontos extremos da fronteira Pareto obtidos por cada algoritmo, ou seja, o ponto mínimo do *makespan* e o ponto mínimo TDT. O valor da distância é capaz de medir a extensão da fronteira Pareto aproximada. O cálculo da distância que tem o maior valor tem a melhor descrição da fronteira Pareto.

Os resultados dos algoritmos para o indicador de distância são apresentados na Tabela 6. A distância média foi aplicada para os grupos de instâncias com o mesmo

Tabela 6 – Resultados do indicador de distância média para os algoritmos ILS/D e MOEA/D

n	m	ILS/D $g = 50$	MOEA/D $g = 50$	MOEA/D $g = 500$	MOEA/D $g = 5000$	MOEA/D $g = 50000$	MOEA/D $g = 500000$
100	20	56,80	**158,21**	65,02	40,94	40,68	39,14
	10	**1516,79**	1306,16	547,06	404,67	530,48	687,66
	5	**10210,12**	8171,41	5221,05	4425,16	5566,94	7011,47
200	20	685,52	**1087,37**	726,33	188,69	107,48	157,88
	10	**7564,68**	7002,06	6286,59	3400,62	2970,61	3518,15
	5	**38015,58**	17757,83	21659,80	13820,86	19290,35	24537,30

número de tarefas e máquinas ($n \times m$), sendo que para cada combinação encontram-se 150 instâncias. Os melhores resultados médios são os valores em negrito.

Na Tabela 6, claramente, pode-se ver que o algoritmo ILS/D obteve melhores resultados para os problemas teste nos grupos 100×10, 100×5, 200×10 e 200×5. Por outro lado, o algoritmo MOEA/D ($g = 50$) apresenta melhores resultados para os problemas teste 100×20 e 200×20. Portanto, os resultados sugerem que mesmo sem apresentar os melhores valores para o indicador de distância para todos os grupos de instâncias, as fronteiras Pareto aproximadas obtidas pelo algoritmo ILS/D possuem melhor extensão para a maioria das instâncias.

5.2.4 Comparação entre os Resultados dos Algoritmos ILS/D, MOEA/D e NSGA-II

Para verificar o desempenho do algoritmo ILS/D, realizou-se a comparação com o algoritmo NSGA-II, o qual também é um algoritmo multiobjetivo baseado em população. O algoritmo NSGA-II é capaz de gerar soluções de boa qualidade para a maioria dos problemas em que é aplicado, sendo estas soluções bem distribuídas na fronteira Pareto aproximada.

O algoritmo NSGA-II foi avaliado nas mesmas 900 instâncias, sendo executado cinco vezes para cada uma delas, e apenas os resultados médios foram considerados. Os parâmetros empregados nos testes do algoritmo NSGA-II foram os seguintes: $g = 50$ e $s = 40$ (correspondentes aos mesmos utilizados nos algoritmos ILS/D e MOEA/D); e taxa de mutação = 0,7 (definida com base em observações empíricas durante a execução do algoritmo). De forma semelhante ao MOEA/D, a realização do cruzamento foi parcialmente baseada no operador de cruzamento para sequenciamento de uma única máquina, apresentado em Sadegheih (2006). Em relação ao operador de mutação, os seis exemplos de movimentos apresentados na Seção 4.5.2.2 foram selecionados aleatoriamente e aplicados na prole (*offspring*).

A Figura 9 mostra o teste *Box-plot of Kruskal-Wallis groups* com nível de confiança de 95% a partir da comparação dos algoritmos ILS/D, MOEA/D e NSGA-II ($g = 50$)

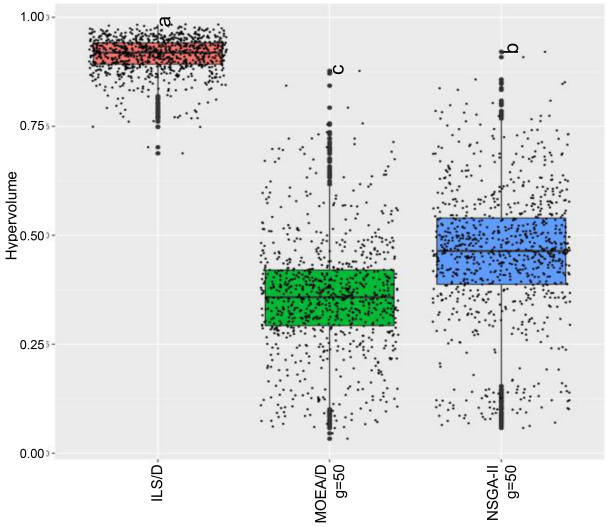


Figura 9 – Resultados do teste *Box-plot of Kruskal-Wallis groups* dos algoritmos ILS/D, MOEA/D e NSGA-II para o indicador HV, de todos os problemas teste.

para todos os problemas de teste. Como o intervalo para o ILS/D é classificado como o grupo a, o intervalo para o MOEA/D é classificado como o grupo c e o intervalo para o NSGA-II é classificado como o grupo b, a média dos algoritmos é significativamente diferente. Ou seja, o algoritmo ILS/D desenvolvido apresentou desempenho superior aos algoritmos MOEA/D e NSGA-II. No entanto, pode-se ver que o algoritmo NSGA-II tem um desempenho melhor do que o algoritmo MOEA/D.

Assim como foi feito, na Seção 5.2.3, o experimento entre os algoritmos ILS/D e MOEA/D, realizou-se também o experimento com o parâmetro g fixo para o ILS/D ($g = 50$) e o parâmetro g variando para o NSGA-II ($g = 50$, $g = 500$, $g = 5000$ e $g = 50000$). A Figura 10 mostra o teste *Box-plot of Kruskal-Wallis groups* com nível de confiança de 95% a partir da comparação dos algoritmos ILS/D e NSGA-II para todos os problemas teste.

Na Figura 10, como o intervalo para o ILS/D ($g = 50$) é classificado como o grupo b, os resultados do teste sugerem que o ILS/D é estatisticamente melhor que o NSGA-II ($g = 50$, $g = 500$ e $g = 5000$), classificados como grupo e, d e c, respectivamente. Entretanto, quando o ILS/D é comparado com o NSGA-II ($g = 50000$), o algoritmo NSGA-II é classificado como grupo a, em uma comparação par-a-par. Assim, o NSGA-II ($g = 50000$) apresenta resultados superiores para todos os problemas teste.

Nota-se que o NSGA-II precisa de mais gerações que o ILS/D para obter um bom desempenho, ou seja, o NSGA-II precisa de 50000 gerações, enquanto o ILS/D precisa de apenas 50 gerações. Por outro lado, os resultados sugerem que o NSGA-II precisa de menos gerações que o MOEA/D para superar o algoritmo ILS/D, ou seja, o MOEA/D precisa de 500.000 gerações para superar o ILS/D.

Como a complexidade computacional do NSGA-II é dada por $\mathcal{O}(Rs^2)$, onde R é o

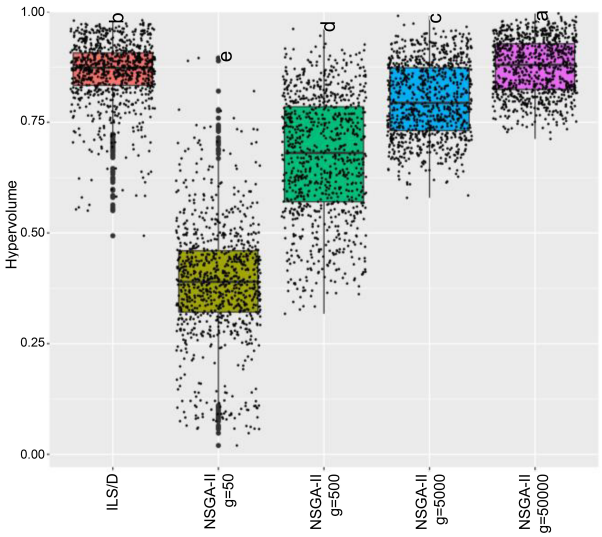


Figura 10 – Resultados do teste *Box-plot of Kruskal-Wallis groups* dos algoritmos ILS/D e NSGA-II para o indicador HV, de todos os problemas teste.

número de objetivos e s é o tamanho da população, o algoritmo MOEA/D ($\mathcal{O}(Rs\eta)$) tem menor complexidade computacional que o NSGA-II a cada geração (ZHANG; LI, 2007). No presente trabalho, entende-se que operações com maior complexidade computacional, no sentido de realizar um maior número de modificações na solução corrente e explorar mais profundamente as soluções vizinhas em cada geração, possibilitam ao NSGA-II obter resultados superiores com menos gerações em comparação ao algoritmo MOEA/D. O mesmo princípio pode ser estendido ao comparar a complexidade computacional do ILS/D com os algoritmos MOEA/D e NSGA-II. Em outras palavras, a maior complexidade de cada geração no algoritmo ILS/D ($\mathcal{O}(Rs\eta\tau)$) permite: (i) uma exploração mais abrangente do espaço de busca em torno de cada indivíduo devido às seis estruturas de vizinhanças realizadas na busca local e ao método de perturbação; e (ii) uma maior diversificação das soluções devido à orientação dos vetores de pesos. Isso, por sua vez, possibilita encontrar resultados superiores em um menor número de gerações.

Os dados do problema, bem como todas as soluções e resultados obtidos, estão disponíveis no site ⟨http://vivian.tmserver.com.br/scheduling/⟩.

5.2.5 Comparação entre os Algoritmos ILS/D e PILS

Assim como foi feito na Seção 5.2.2, a calibração dos parâmetros do algoritmo PILS foi realizada com um conjunto independente de 36 instâncias e cada instância foi executada 5 vezes.

O algoritmo PILS possui três parâmetros de entrada a serem definidos: τ: controla o tempo máximo de execução do algoritmo; t: define o tamanho da perturbação; ϕ: define o número de soluções criadas pelo procedimento construtivo.

No algoritmo ILS/D, o método ILS é executado para s subproblemas e g gerações, com o parâmetro τ para o ILS definido para o tempo de CPU de $0,08 \times \frac{n}{m}$ segundos. Portanto, para utilizar o mesmo orçamento computacional para o algoritmo PILS, foi necessário incluir esses valores ao tempo máximo de execução do algoritmo PILS. Sendo assim, o parâmetro τ para o algoritmo PILS foi definido para o tempo de CPU: $s \times g \times 0,08 \times \frac{n}{m}$ segundos. Na calibração dos parâmetros do algoritmo ILS/D foi definido que $s = 40$ e $g = 50$. Assim, cada execução do algoritmo PILS gastou $50 \times 40 \times 0.08 \times \frac{n}{m}$ segundos.

Os três valores testados para o parâmetro ϕ são {20; 40; 60} e os dois valores testados para o parâmetro t são $\{0,7 \times m;\ 0,9 \times m\}$. Observe que há um total de 6 combinações de parâmetros a serem avaliadas. Sendo que foram realizadas 5 execuções para as 36 instâncias, totalizando 1080 execuções.

As seis configurações testadas para o algoritmo PILS não apresentam diferenças significativas quando comparadas entre si, apresentando as seguintes médias de hipervolume: 0,726, 0,700, 0,701, 0,711, 0,723, 0,720, respectivamente. Embora as configurações sejam estatisticamente iguais, optamos pela configuração 05, que apresenta alto valor para a média do hipervolume (0,723) e menor desvio padrão (0,12). Portanto, para os experimentos finais do PILS, utilizamos os parâmetros utilizados na configuração 05: $\tau = 50 \times 40 \times 0,08 \times \frac{n}{m}$ segundos, $\phi = 60$ e $t = 0,7 \times m$.

Os dois algoritmos foram testados em 900 instâncias propostas por Ruiz-Torres, Paletta e Perez-Roman (2015) e foram executados 5 vezes com o mesmo tempo de CPU como critério de parada.

A Tabela 7 mostra o HV médio encontrado pelos algoritmos comparados para todas as instâncias. As médias de HV são agrupados pelo tamanho da instância ($n \times m \times d_{jk} \times CR$), sendo que para cada combinação encontram-se 25 instâncias, que foram submetidas a 5 execuções, totalizando 125 repetições. Os valores em negrito representam os melhores resultados. Claramente, é possível ver que o algoritmo ILS/D apresenta os melhores resultados para os problemas de teste 100×5, 200×20, 200×10 e 200×5. Por outro lado, o algoritmo PILS apresenta melhores resultados para os problemas de teste 100×10. E para os problemas de teste 100×20, os dois algoritmos mostram resultados semelhantes. Percebe-se que o ILS/D teve menos tempo para convergir para estes problemas teste.

O algoritmo ILS/D tem um bom desempenho para resolver os problemas mais difíceis. Portanto, os resultados sugerem que o algoritmo ILS/D tem um desempenho melhor do que o algoritmo PILS, porque o algoritmo ILS/D encontrou o melhor resultado na maioria das instâncias. De acordo com a Tabela 7, o ILS/D obteve o melhor desempenho em 80% dos casos.

Verificou-se que os dados não seguem uma distribuição normal usando o teste

Tabela 7 – Resultados da média do indicador HV para os algoritmos ILS/D e PILS.

n	m	d_{jk}	CR	ILS/D	PILS	n	m	d_{jk}	CR	ILS/D	PILS
100	20	[1%, 5%]	2	**0.6309**	0.5802	200	20	[1%, 5%]	2	**0.7975**	0.7731
			3.5	**0.6544**	0.6293				3.5	**0.8278**	0.8035
			5	0.6267	**0.6529**				5	**0.7149**	0.6880
		[5%, 10%]	2	**0.6476**	0.6331			[5%, 10%]	2	**0.8666**	0.8150
			3.5	**0.7062**	0.6824				3.5	**0.7800**	0.7379
			5	0.6533	**0.7085**				5	**0.7158**	0.6527
	10	[1%, 5%]	2	**0.8495**	0.7972		10	[1%, 5%]	2	0.9175	**0.9218**
			3.5	**0.7820**	0.7768				3.5	**0.6544**	0.6458
			5	0.7357	**0.7724**				5	**0.6645**	0.6213
		[5%, 10%]	2	0.8552	**0.8703**			[5%, 10%]	2	**0.7577**	0.7548
			3.5	0.7352	**0.7874**				3.5	**0.7616**	0.7141
			5	0.7334	**0.7773**				5	**0.7544**	0.7302
	5	[1%, 5%]	2	**0.7924**	0.7363		5	[1%, 5%]	2	**0.7614**	0.6387
			3.5	**0.7521**	0.7225				3.5	**0.7427**	0.6267
			5	**0.7468**	0.7038				5	**0.7261**	0.6070
		[5%, 10%]	2	**0.7604**	0.7356			[5%, 10%]	2	**0.7948**	0.7372
			3.5	**0.7343**	0.7199				3.5	**0.8011**	0.7544
			5	**0.7316**	0.7153				5	**0.7989**	0.7575

Shapiro-Wilk. Para validar os resultados obtidos pelos dois algoritmos e verificar se as diferenças observadas são estatisticamente significativas, foi aplicado o teste não paramétrico *Kruskal-Wallis*, para os grupos de instâncias de tamanho ($n \times m$), sendo que para cada combinação encontram-se 150 instâncias e apenas o resultado médio foi considerado.

O valor obtido para o *p-value* no teste *Kruskal-Wallis* foi inferior a 0,05 para os problemas de teste: 100×5, 200×20, 200×10 e 200×5. Estes resultados sugerem que há diferença estatística entre os resultados do indicador HV para os algoritmos comparados com nível de confiança de 95% (limite = 0,05). Por outro lado, para os problemas de teste: 100×20 e 100×10, o resultado *p-value* obtido no teste *Kruskal-Wallis* foi maior que 0,05, o que mostra que não há diferença entre os algoritmos para estes tipos de instâncias.

Para identificar diferenças significativas entre os grupos dos algoritmos, e indicar quais pares específicos de grupos diferem entre si, foi realizado o teste *Box-plot of Kruskal-Wallis groups* com nível de confiança de 95%, juntamente com os resultados do teste *post hoc*. A análise do teste pode ser vista na Figura 11, a qual mostra a comparação dos algoritmos ILS/D e PILS para os problemas de teste: 100×20 (Figura 11-a), 100×10 (Figura 11-b), 100×5 (Figura 11-c), 200×20 (Figura 11-d), 200×10 (Figura 11-e) e 200×5 (Figura 11-f).

Na Figura 11 (c), (d), (e) e (f), uma vez que o intervalo para o algoritmo ILS/D é classificado como o grupo a e o intervalo para o algoritmo PILS é classificado como o grupo b, a média do algoritmo PILS é significativamente diferente. Ou seja, o algoritmo ILS/D apresentou desempenho superior ao algoritmo PILS. No entanto, para os problemas de teste: 100×20 e 100×10, pode-se ver na Figura 11 (a) e (b), que os algoritmos não apresentam diferenças significativas quando estas foram comparadas entre si, uma vez que

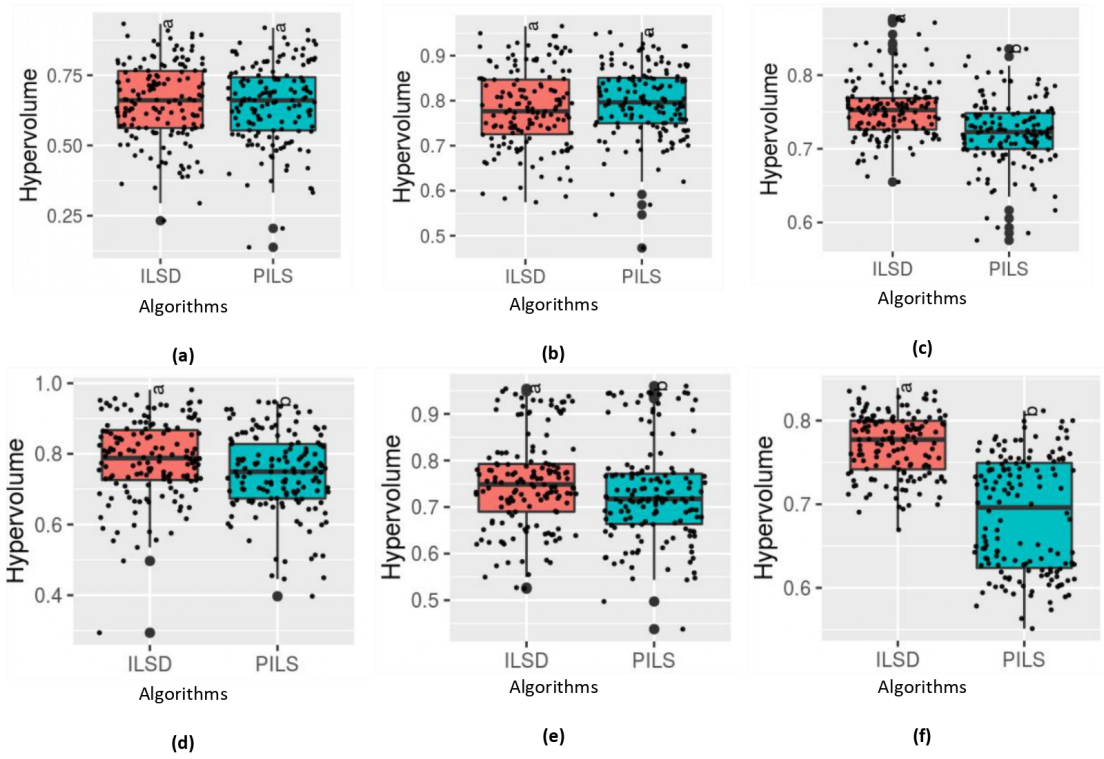


Figura 11 – Resultados do teste *Box-plot of Kruskal-Wallis groups* dos algoritmos ILS/D e PILS para o indicador HV, dos problemas teste $n \times m$.

todas as configurações fazem parte do grupo a.

Em geral, o resultado do teste *Box-plot of Kruskal-Wallis groups* sugere que ILS/D é estatisticamente melhor do que o outro algoritmo em uma comparação de par-a-par.

Uma análise gráfica da convergência dos algoritmos foi realizada. Para isso, seis instâncias ($n \times m$) foram selecionadas aleatoriamente para ilustrar os resultados. A Figura 12 mostra os resultados da melhor execução entre as cinco execuções das instâncias selecionadas. A Figura 12 sugere que o algoritmo ILS/D tem uma boa convergência para a fronteira Pareto, considerando o ambiente experimental para os problemas de teste: 100×20 (Figura 12-a), 100×10 (Figura 12-b), 100×5 (Figura 12-c), 200×20 (Figura 12-d), 200×10 (Figura 12-e) e 200×5 (Figura 12-f). Além disso, a Figura 12 sugere que as fronteiras Pareto aproximadas obtidas por ILS/D têm maior extensão.

Por outro lado, a Figura 12 mostra que uma limitação do algoritmo PILS é que não há controle da diversificação na fronteira Pareto aproximada. Embora o algoritmo PILS obtenha um melhor valor de hipervolume para algumas instâncias, pode-se verificar que há uma pobre extensão da fronteira Pareto aproximada. Além disso, a Figura 12 mostra que uma marca distintiva do algoritmo ILS/D é a setorização da busca na fronteira Pareto aproximada, provavelmente por se beneficiar do paralelismo implícito de uma busca

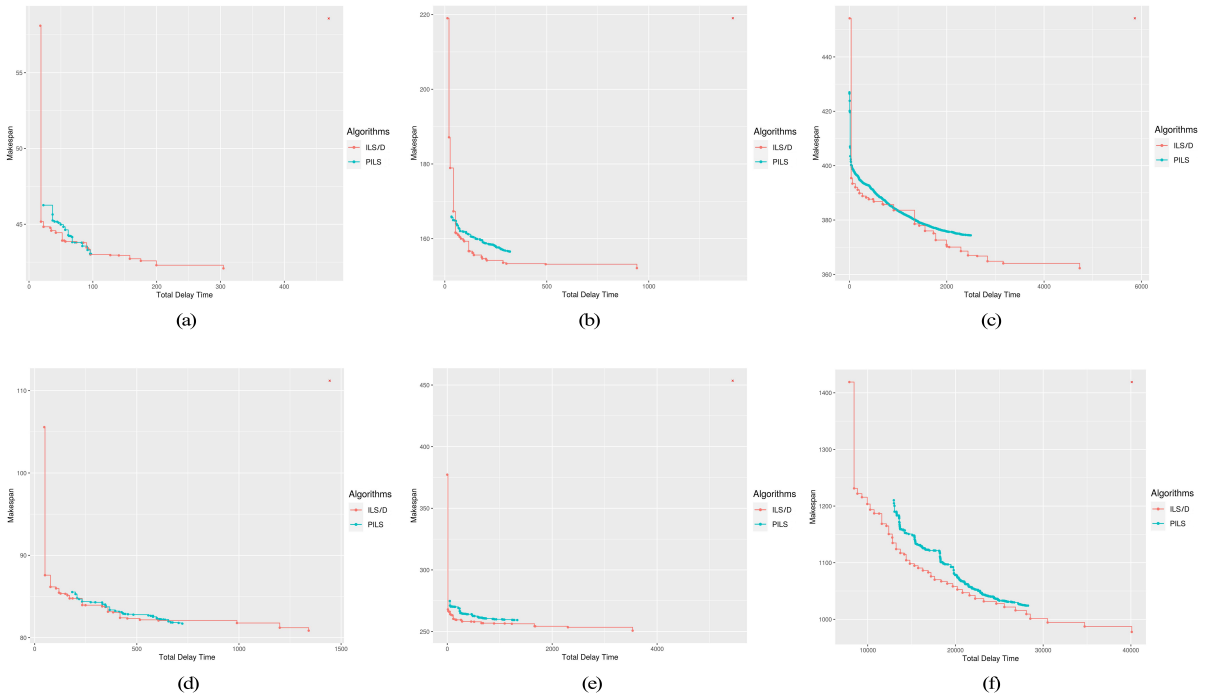


Figura 12 – Fronteiras Pareto aproximadas obtidas com os algoritmos ILS/D e PILS para os problemas teste $n \times m$.

e decomposição baseada na população.

Um segundo indicador também é usado para avaliar a qualidade das fronteiras Pareto aproximadas obtidas pelos algoritmos. Este indicador é o cálculo do valor da distância Euclidiana entre os dois pontos extremos da fronteira Pareto obtidas por cada algoritmo, ou seja, o ponto mínimo do *makespan* e o ponto mínimo do TDT. O valor da distância é capaz de medir a extensão da fronteira Pareto aproximada. O cálculo da distância que possui o maior valor tem a melhor descrição da fronteira Pareto.

Os resultados dos algoritmos para o indicador de distância são apresentados na Tabela 8. A distância média foi aplicada para os grupos de instâncias com o mesmo número de tarefas e máquinas ($n \times m$), sendo que para cada combinação encontram-se 150 instâncias, que foram submetidas a 5 execuções, totalizando 750 repetições. Além disso, a tabela mostra os valores de hipervolume para os grupos de instâncias ($n \times m$). Os melhores resultados médios da Tabela 8 são valores em negrito. Claramente, pode-se ver que o algoritmo ILS/D obteve melhores resultados em todos os casos para o indicador de distância e apresenta os melhores resultados para quase todos os problemas de teste considerando o indicador de hipervolume. Portanto, os resultados sugerem que mesmo sem apresentar os melhores valores para o indicador de hipervolume para todos os grupos de instâncias, as fronteiras Pareto aproximadas obtidas pelo algoritmo ILS/D apresentam melhor extensão.

Tabela 8 – Resultados da média dos indicadores HV e distância para os algoritmos ILS/D e PILS.

		Hipervolume		Distância	
n	m	ILS/D	PILS	ILS/D	PILS
100	20	**0,6532**	0,6477	**56,80**	33,96
	10	0,7818	**0,7969**	**1516,79**	858,63
	5	**0,7529**	0,7222	**10210,12**	7458,92
200	20	**0,7838**	0,7450	**685,52**	413,25
	10	**0,7517**	0,7313	**7564,68**	5224,05
	5	**0,7708**	0,6869	**38015,58**	24203,30

5.2.6 Validação dos Algoritmos

O principal objetivo desta seção é validar os resultados obtidos dos algoritmos ILS/D, MOEA/D e NSGA-II para o problema abordado. Para isso, estes algoritmos são executados para quarenta e oito instâncias de pequeno porte propostas neste trabalho. Essas instâncias usam combinações dos seguintes valores n e m: $n \in \{6, 7\}$ e $m \in \{2, 3, 4\}$, adicionalmente, $n \in \{8\}$ e $m \in \{2, 3\}$. Ao combinar esses parâmetros com os outros parâmetros (p_{jk}, d_{jk}, CR), explicados na Seção 5.2.1, foram geradas 48 instâncias no total.

Para encontrar os resultados exatos das instâncias de pequeno porte, foi utilizada a técnica de enumeração. Isso se deve à impossibilidade de executar o modelo devido à sua não linearidade, como explicado na Seção 3.2.2. Assim, os resultados dos algoritmos são comparados com os valores ótimos obtidos pela técnica de enumeração. Contando todos os pontos exatos da fronteira Pareto-ótimo obtida pela técnica de enumeração, verificou-se que, em média, os algoritmos MOEA/D, ILS/D e NSGA-II encontram 88%, 92% e 97% pontos iguais, respectivamente.

A Figura 13 mostra os resultados obtidos pela técnica de enumeração e pelos algoritmos MOEA/D, ILS/D e NSGA-II para oito instâncias selecionadas aleatoriamente de cada combinação n e m. A partir da ilustração, pode-se ver que os resultados alcançados pelos algoritmos são semelhantes à técnica de enumeração.

Esses resultados confirmam a validade dos algoritmos ILS/D, MOEA/D e NSGA-II para o problema, mostrando que eles são capazes de convergir para soluções corretas da fronteira Pareto-ótimo.

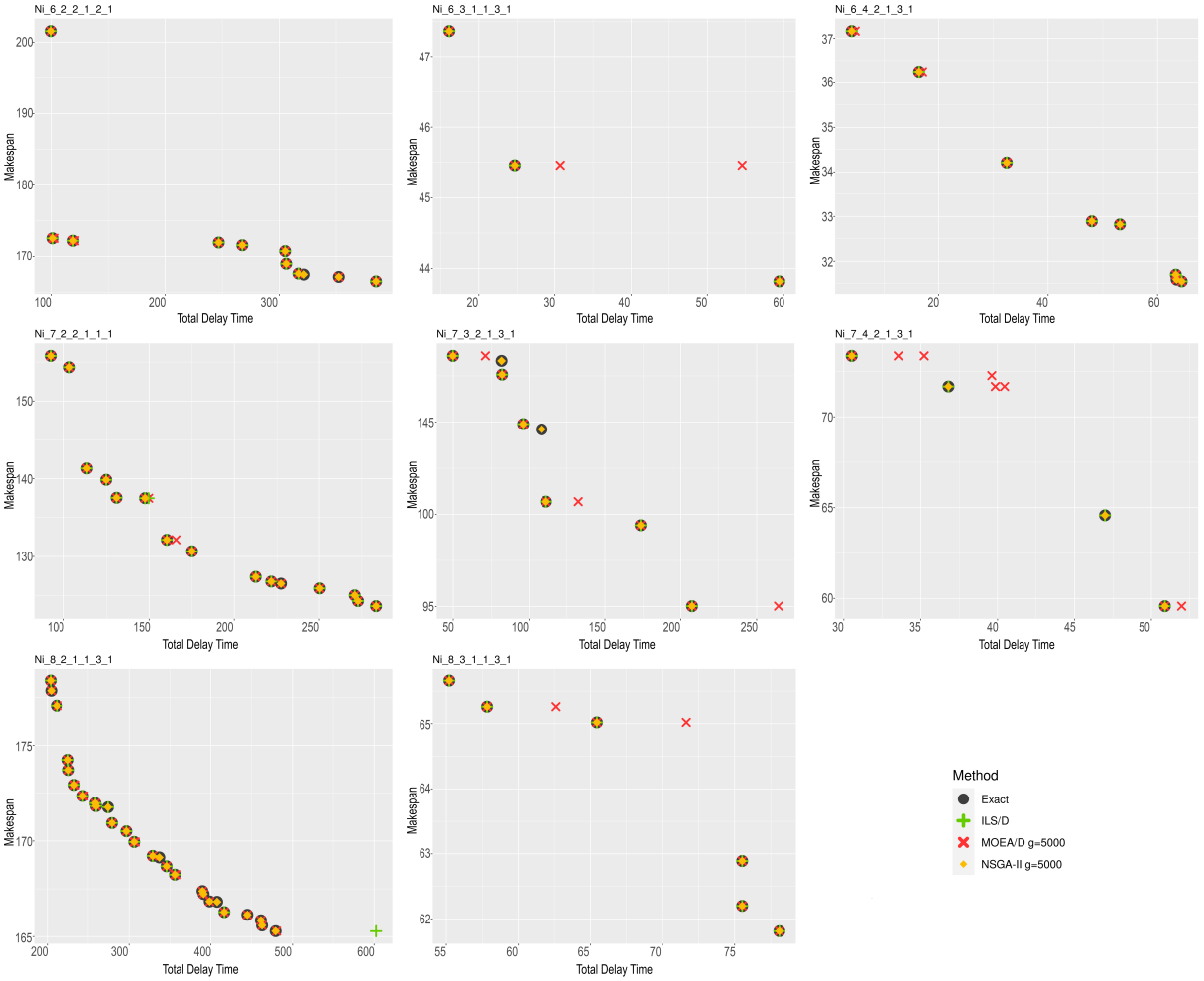


Figura 13 – Fronteira Pareto-ótimo (círculos pretos) e as fronteiras retornadas pelos algoritmos ILS/D (‘+’ verde), MOEA/D (‘×’ vermelho) e NSGA-II (retângulo amarelo).

5.3 Experimentos Computacionais II

Esta seção apresenta os experimentos computacionais que foram realizados usando o algoritmo ILS/D original (sem nenhuma estratégia de manutenção) e levando em consideração as três estratégias de manutenção. Os algoritmos heurísticos foram desenvolvidos em C++ e executados em cinco computadores independentes (software e hardware), cada um com Intel® Core TM i7-870, CPU 2,93 GHz, 8 GB de RAM, rodando Ubuntu 16.04.5 LTS. Os valores dos parâmetros utilizados no algoritmo ILS/D são os mesmos que foram calibrados na Seção 5.2.2. O desempenho de cada algoritmo é medido pelo cálculo da métrica HV.

5.3.1 Instâncias do Problema

Os algoritmos foram testados em 720 instâncias geradas neste trabalho, as quais estão disponíveis em ⟨http://vivian.tmserver.com.br/scheduling_maintenance_jobs⟩. As

instâncias foram geradas baseadas nos artigos de Ruiz-Torres, Paletta e Perez-Roman (2015) e Ruiz-Torres, Paletta e M'Hallah (2017).

O tamanho de uma instância é definido pelo número de tarefas (n) e pelo número de máquinas (m). As instâncias usam combinações dos seguintes valores n e m: $n \in \{100, 200\}$ e $m \in \{5, 10, 20\}$. Em todas as instâncias, os tempos de processamento p_{jk} foram gerados aleatoriamente, com distribuição uniforme, no intervalo: [1; 100]. Os efeitos de deterioração (d_{jk}) também foram gerados aleatoriamente, com distribuição uniforme em dois intervalos: [1%, 5%] e [5%, 10%].

A taxa de congestionamento (CR) é usada para estabelecer o rigor geral da data de vencimento, um CR mais alto resultaria em um número maior de atrasos nas entregas das tarefas. Os valores de CR das datas de entrega e_j para cada tarefa j são dados por: $CR \in \{2, 3.5, 5\}$. E a unidade de tempo (u_k) de uma tarefa de manutenção na máquina k é distribuída uniformemente em dois intervalos: [1, 3] e [1, 9].

Para cada combinação de n, m, intervalo de tempo de processamento, intervalo de efeito de deterioração, taxa de congestionamento de datas de vencimento e tempo de manutenção, obteve-se 72 combinações. Para cada combinação foram geradas 10 instâncias, totalizando 720 instâncias.

5.3.2 Resultados Computacionais do ILS/D sem Manutenção e com Estratégias de Manutenção

Nesta seção, o algoritmo ILS/D sem manutenção (nomeado, None) e o algoritmo ILS/D considerando as três diferentes estratégias de manutenção (MS1, MS2 e MS3) são avaliados em 720 instâncias. Os algoritmos foram executados cinco vezes para cada instância e apenas os resultados médios são considerados.

A Tabela 9 apresenta os resultados médios obtidos para a métrica HV. Os HV médios são agrupados por tamanho de instância ($n \times m \times d_{jk} \times CR \times u_k$), sendo que em cada combinação encontram-se 10 instâncias, que foram executadas 5 vezes cada, totalizando 50 repetições. Os valores em negrito representam os melhores resultados.

De acordo com a Tabela 9 para o problema de teste de 100×20, todos os algoritmos obtiveram os melhores resultados para algum grupo de instâncias. O algoritmo MS1 obteve os melhores resultados para os problemas teste 100×10. O algoritmo MS3 obteve os melhores resultados para a maioria dos problemas de teste 200×20 e 200×10. Os algoritmos MS2 e MS3 obtiveram os melhores resultados para os problemas de teste 100×5 e 200×5, ou seja, para esses problemas de teste, MS2 e MS3 apresentaram resultados semelhantes. Entretanto, o algoritmo ILS/D original (None), não apresenta melhores resultados quando comparado a nenhuma das três estratégias. Assim, os resultados sugerem que a aplicação de estratégias de manutenção permite encontrar melhores resultados para o problema

Tabela 9 – Resultados da média do indicador HV para os algoritmos MS1, MS2, MS3 e None.

n	m	d_{jk}	CR	u_k	MS1	MS2	MS3	None
100	10	[1%,5%]	2	[1,3]	**0,7212**	0,6852	0,6706	0,6654
				[1,9]	0,6590	0,6657	**0,7121**	0,6796
			3,5	[1,3]	0,7072	**0,7172**	0,7094	0,7019
				[1,9]	0,7252	0,6950	**0,7405**	0,7027
			5	[1,3]	0,7192	0,6679	**0,7204**	0,7015
				[1,9]	0,7058	0,6732	0,6885	**0,7087**
		[5%, 10%]	2	[1,3]	**0,7428**	0,5935	0,6778	0,6599
				[1,9]	0,7134	0,6827	**0,7364**	0,7022
			3,5	[1,3]	**0,7303**	0,5660	0,6885	0,6509
				[1,9]	**0,7595**	0,6586	0,7205	0,6997
			5	[1,3]	**0,7789**	0,6522	0,7097	0,7327
				[1,9]	0,7520	0,7047	0,7571	**0,7706**
	20	[1%,5%]	2	[1,3]	**0,9275**	0,8482	0,9088	0,8605
				[1,9]	**0,9065**	0,8397	0,8986	0,8960
			3,5	[1,3]	0,8346	0,7722	**0,8495**	0,6556
				[1,9]	0,7971	0,7287	**0,8148**	0,7413
			5	[1,3]	**0,8173**	0,6651	0,8126	0,5702
				[1,9]	**0,8164**	0,7358	0,8005	0,7100
		[5%, 10%]	2	[1,3]	**0,9359**	0,8809	0,9300	0,6965
				[1,9]	0,8947	0,8635	**0,9164**	0,7556
			3,5	[1,3]	**0,9215**	0,8555	0,8887	0,4789
				[1,9]	0,8618	0,8226	**0,8738**	0,5168
			5	[1,3]	**0,9269**	0,8727	0,9199	0,4274
				[1,9]	**0,8971**	0,8406	0,8817	0,4538
	5	[1%,5%]	2	[1,3]	0,9456	**0,9602**	0,9534	0,5692
				[1,9]	0,9273	**0,9488**	0,9480	0,6252
			3,5	[1,3]	0,9440	0,9621	**0,9688**	0,4904
				[1,9]	0,9317	**0,9559**	**0,9559**	0,5360
			5	[1,3]	0,9556	0,9582	**0,9626**	0,4855
				[1,9]	0,9340	0,9306	**0,9593**	0,5273
		[5%, 10%]	2	[1,3]	0,9489	**0,9883**	0,9861	0,3704
				[1,9]	0,9371	**0,9878**	0,9838	0,3902
			3,5	[1,3]	0,9245	**0,9808**	0,9796	0,3089
				[1,9]	0,9155	0,9832	**0,9853**	0,3177
			5	[1,3]	0,9312	**0,9838**	0,9812	0,2788
				[1,9]	0,9191	**0,9803**	0,9799	0,2985
200	10	[1%,5%]	2	[1,3]	0,8708	0,7659	**0,9035**	0,8961
				[1,9]	**0,8419**	0,7127	0,8386	0,8149
			3,5	[1,3]	0,8624	0,8109	**0,8929**	0,8558
				[1,9]	0,8755	0,8483	0,8877	**0,8954**
			5	[1,3]	0,7466	0,7233	**0,7499**	0,7275
				[1,9]	0,7480	0,6630	**0,7547**	0,7479
		[5%, 10%]	2	[1,3]	0,8415	0,8550	**0,9530**	0,7712
				[1,9]	0,8471	0,8695	**0,9466**	0,8480
			3,5	[1,3]	0,8693	0,8414	**0,9180**	0,6231
				[1,9]	0,7858	0,8048	**0,8670**	0,7091
			5	[1,3]	0,7629	0,7561	**0,8851**	0,4305
				[1,9]	0,6608	0,7210	**0,7785**	0,5530
	20	[1%,5%]	2	[1,3]	0,9105	0,9240	**0,9646**	0,7733
				[1,9]	0,8904	0,9150	**0,9444**	0,8062
			3,5	[1,3]	0,8620	**0,8793**	0,8792	0,3594
				[1,9]	0,8085	0,8635	**0,8696**	0,4414
			5	[1,3]	0,8585	0,8857	**0,9076**	0,3490
				[1,9]	0,8466	0,8763	**0,8892**	0,4314
		[5%, 10%]	2	[1,3]	0,9437	0,9470	**0,9758**	0,4027
				[1,9]	0,9034	0,9474	**0,9614**	0,4193
			3,5	[1,3]	0,9051	0,9568	**0,9712**	0,2911
				[1,9]	0,8482	**0,9561**	0,9494	0,3135
			5	[1,3]	0,8792	0,9564	**0,9570**	0,2547
				[1,9]	0,8449	**0,9442**	0,9312	0,2771
	5	[1%,5%]	2	[1,3]	0,9256	**0,9863**	0,9778	0,3761
				[1,9]	0,9074	**0,9841**	0,9643	0,3820
			3,5	[1,3]	0,8947	0,9735	**0,9753**	0,3101
				[1,9]	0,8733	**0,9718**	0,9643	0,3237
			5	[1,3]	0,9109	0,9813	**0,9834**	0,3324
				[1,9]	0,8773	0,9669	**0,9714**	0,3187
		[5%, 10%]	2	[1,3]	0,8511	0,9952	**0,9963**	0,1591
				[1,9]	0,8416	**0,9952**	0,9916	0,1700
			3,5	[1,3]	0,8125	0,9920	**0,9929**	0,1435
				[1,9]	0,7882	**0,9915**	0,9858	0,1452
			5	[1,3]	0,7970	0,9897	**0,9940**	0,1421
				[1,9]	0,7652	**0,9916**	0,9829	0,1431

abordado e que o MS3 tem um desempenho melhor do que as estratégias MS1 e MS2 para a maioria dos problemas de teste.

De acordo com os resultados obtidos nas 5 execuções dos algoritmos e mostrados na Tabela 9, o algoritmo MS3 obteve a melhor média HV em 52,8% das instâncias; o algoritmo MS2 obteve a melhor média HV em 25,0% das instâncias; o algoritmo MS1 obteve a melhor média HV em 19,4% das instâncias, e o algoritmo None obteve a melhor média HV em 4,2% das instâncias.

Os três pressupostos dos testes paramétricos foram verificados para validar os resultados obtidos pelos algoritmos: normalidade, igualdade de variância e resíduos independentes. No entanto, verificou-se que os dados não seguem uma distribuição normal usando o teste *Shapiro-Wilk* (SHAPIRO; WILK, 1965), sendo assim, foi necessário utilizar testes não paramétricos. O teste *Kruskal-Wallis* permite a comparação de três ou mais grupos em amostras independentes. O teste *Kruskal-Wallis* é o método não paramétrico alternativo à ANOVA para um fator, sendo utilizado nos casos em que os pressupostos exigidos pela ANOVA não são atendidos (KRUSKAL; WALLIS, 1952).

Para verificar se as diferenças observadas são estatisticamente significativas, aplicou-se o teste não paramétrico *Kruskal-Wallis* para os grupos de tamanho de instância ($n \times m$), sendo que para cada combinação encontram-se 120 instâncias, ou seja , 120 amostras que foram submetidas a 5 execuções, totalizando 600 repetições e apenas o resultado médio foi considerado. O valor obtido para o p-value no teste *Kruskal-Wallis* foi menor que 0,05 para todos os problemas teste. Os resultados sugerem que há uma diferença estatística entre os resultados do indicador HV para os algoritmos comparados com nível de confiança de 95% (limiar = 0,05).

Para identificar diferenças significativas entre os grupos dos algoritmos, e indicar quais pares específicos de grupos diferem entre si, foi realizado o teste *Box-plot of Kruskal-Wallis groups* com nível de confiança de 95%, juntamente com os resultados do teste *post hoc*. A Figura 14 mostra esta análise a partir da comparação do algoritmo ILS/D sem manutenção (None) e algoritmos ILS/D com três estratégias de manutenção MS1, MS2 e MS3 para os problemas de teste 100×20 (Figura 14-a), 100×10 (Figura 14-b), 100×5 (Figura 14-c), 200×20 (Figura 14-d), 200×10 (Figura 14-e) e 200×5 (Figura 14-f).

Na Figura 14 (a), para os problemas de teste 100×20, percebe-se que o algoritmo MS1 apresenta um comportamento parcialmente melhor que o algoritmo MS3 e um desempenho melhor que os demais algoritmos, uma vez que MS1 é único classificado apenas como o grupo a. Além disso, o algoritmo MS3 tem desempenho semelhante aos algoritmos MS1 e None, já que é classificado como o grupo ab. E o algoritmo MS2 (grupo c) tem um desempenho inferior a todos os outros algoritmos.

Na Figura 14 (b), uma vez que o intervalo para os algoritmos MS1 e MS3

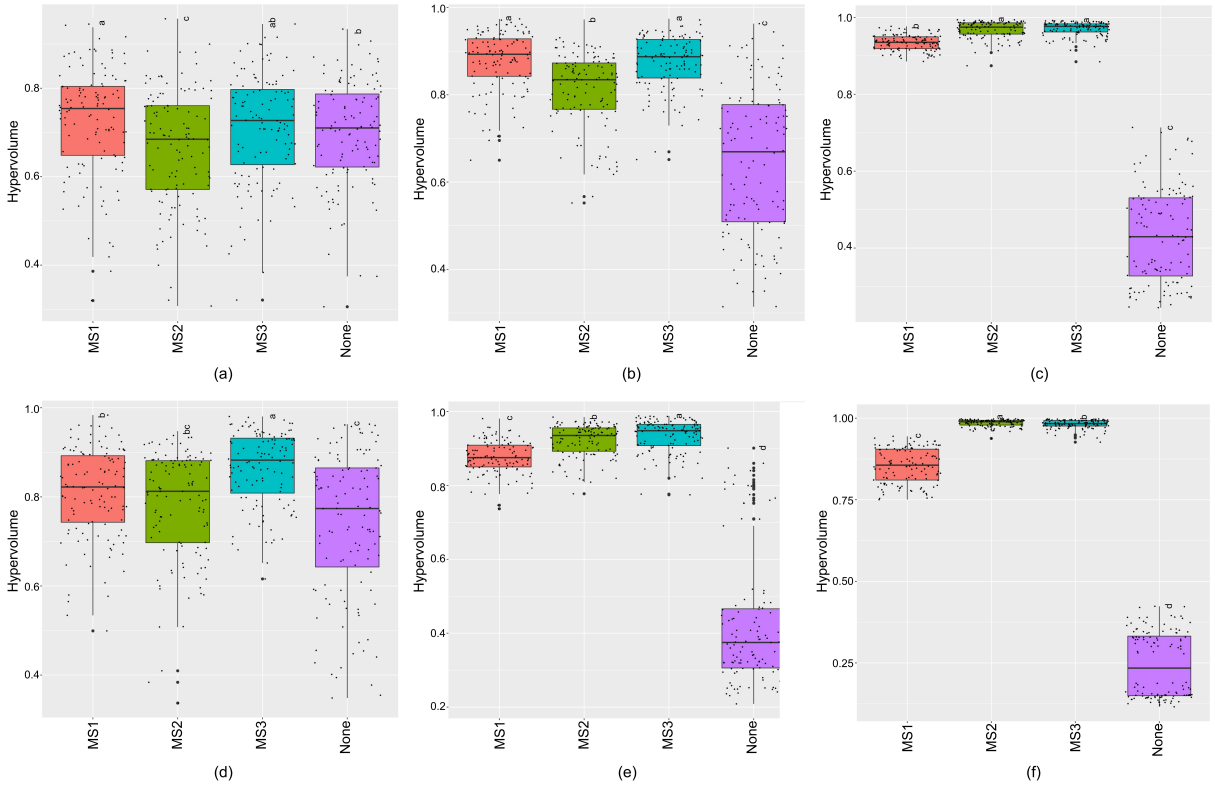


Figura 14 – Resultados do teste *Box-plot of Kruskal-Wallis groups* para os algoritmos MS1, MS2, MS3, e None para o indicador HV dos problemas teste $n \times m$.

são classificados como o grupo a e o intervalo para os algoritmos MS2 e None são classificados como o grupo b e c, respectivamente, as médias dos algoritmos MS1 e MS3 são significativamente diferentes. Ou seja, apresentaram desempenho superior, para os problemas de teste 100×10.

Na Figura 14 (c), o intervalo para os algoritmos MS2 e MS3 é classificado como o grupo a, o intervalo para o algoritmo MS1 é classificado como o grupo b, e o intervalo para o algoritmo None é classificado como o grupo c. Assim, os algoritmos MS2 e MS3 apresentaram desempenho superior, para os problemas de teste 100×5.

Na Figura 14 (d), pode-se ver que o intervalo para o algoritmo MS3 é classificado como o grupo a para os problemas de teste 200×20, o intervalo para o MS1 é classificado como o grupo b, o intervalo para o algoritmo None é classificado como grupo c, e o MS2 (grupo bc) é classificado no mesmo grupo dos algoritmos MS1 e None. Assim, o algoritmo MS3 apresentou desempenho superior.

Na Figura 14 (e), pode-se ver que o intervalo para o algoritmo MS3 é classificado como o grupo a, e o intervalo para os algoritmos MS2, MS1 e None são classificados como o grupo grupos b, c e d, respectivamente. Ou seja, o algoritmo MS3 apresenta um desempenho melhor que os outros algoritmos quando comparados entre si, para os problemas de teste 200×10.

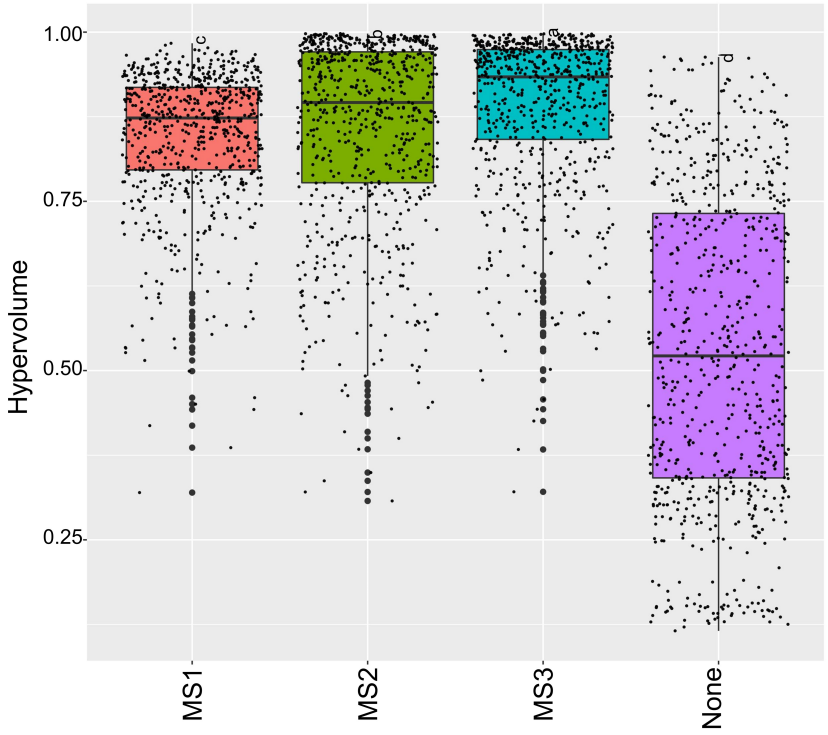


Figura 15 – Resultados do teste *Box-plot of Kruskal-Wallis groups* para os algoritmos MS1, MS2, MS3, e None para o indicador HV de todos os problemas teste.

Na Figura 14 (f), o intervalo para os algoritmos MS2, MS3, MS1 e None são classificados como o grupo a, b, c e d, respectivamente . Assim, o algoritmo MS2 apresentou desempenho superior para os problemas de teste 200×5.

Em geral, os resultados do teste *Box-plot of Kruskal-Wallis groups* sugerem que o algoritmo MS3 é estatisticamente melhor que os outros algoritmos, em uma comparação par-a-par, porque é classificado como o grupo a para a maioria dos problemas de teste. Embora o algoritmo MS3 obtenha a mesma classificação que outros algoritmos na maioria dos casos, ele apresenta rendimento inferior do que outro algoritmo em apenas um problema de teste (200×5). Por outro lado, os resultados sugerem que o algoritmo None apresenta o pior desempenho em praticamente todos os problemas de teste. A exceção está no problema de teste 100×20, onde obtém classificação superior ao algoritmo MS2. Essa análise pode ser confirmada na Figura 15.

A Figura 15 mostra o teste *Box-plot of Kruskal-Wallis groups* com nível de confiança de 95% a partir da comparação dos algoritmos MS1, MS2, MS3 e None para todos os problemas de teste, ou seja , para as 5 execuções das 720 instâncias. Como o intervalo para o algoritmo MS3 é classificado como o grupo a, os intervalos para os outros algoritmos MS2, MS1 e None são classificados como o grupo b, c e d, respectivamente, a média dos algoritmos é significativamente diferente. Assim, o algoritmo MS3 apresentou desempenho melhor, superando os demais algoritmos, em uma comparação par-a-par. Além disso, pode-se ver que o algoritmo MS2 possui rendimento maior do que o algoritmo

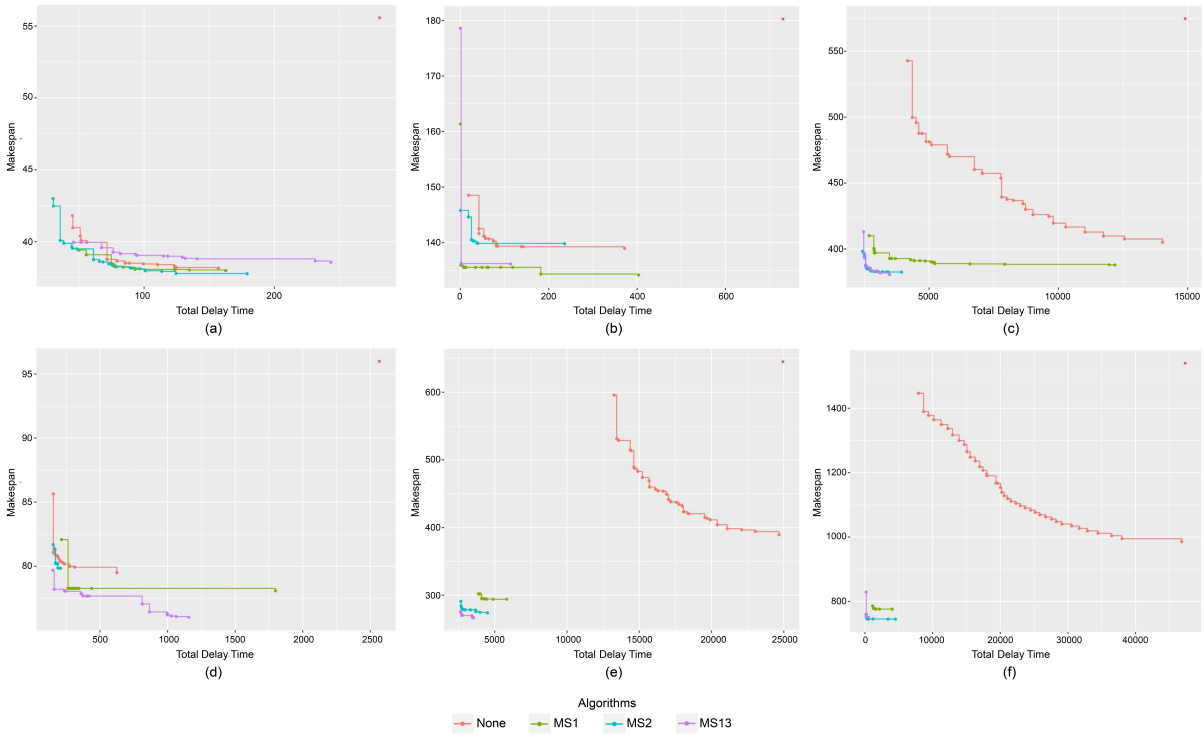


Figura 16 – Fronteiras Pareto aproximadas obtidas com os algoritmos MS1, MS2, MS3, e None para os problemas teste $n \times m$.

MS1. E todos os algoritmos têm melhor desempenho do que o algoritmo None, pois, este é classificado como o grupo d.

Analisando de forma mais abrangente os resultados do ILS/D com diferentes estratégias de manutenção e sem nenhuma, torna-se evidente o impacto e as vantagens do uso das tarefas de manutenção no problema de sequenciamento em questão. Em todos os resultados apresentados nas Figuras 14 e 15, os métodos com estratégias de manutenção possuem um valor de HV maior do que aqueles sem manutenção, e essas diferenças são estatisticamente significativas. Além disso, é possível observar que, mesmo que essas tarefas de manutenção tenham um tempo de execução e, de certa forma, interrompam a produção, esse tempo é compensado pela redução do *makespan* e do atraso total de entrega das tarefas.

Foi realizada uma análise gráfica da convergência do algoritmo. Seis instâncias ($n \times m$) foram selecionadas aleatoriamente para ilustrar os resultados. A Figura 16 mostra os resultados da melhor execução entre as cinco execuções das instâncias selecionadas. A figura 16 sugere que os algoritmos MS1, MS2 e MS3 têm uma boa convergência para a fronteira Pareto aproximada, considerando o ambiente experimental para os problemas de teste 100×20 (Figura 16-a), 100×10 (Figura 16-b), 100×5 (Figura 16-c), 200×20 (Figura 16-d), 200×10 (Figura 16-e) e 200×5 (Figura 16-f).

Por outro lado, a Figura 16 indica que as fronteiras Pareto aproximadas obtidas pelo algoritmo None têm baixa convergência. Na maioria dos casos, este algoritmo não

atinge um alto valor para o indicador de hipervolume.

Os dados do problema, todas as soluções e os resultados obtidos estão disponíveis no site ⟨http://vivian.tmserver.com.br/scheduling_maintenance_jobs⟩.

5.3.3 Discussão

Durante os experimentos realizados, observou-se um efeito positivo do sequenciamento das tarefas de manutenção. Como se observa que a manutenção de equipamentos é benéfica em sistemas reais de manufatura, esse efeito já era esperado. Além disso, o Lema 1 estabelecido por Ruiz-Torres, Paletta e M'Hallah (2017) prova que as operações de manutenção reduzem o tempo de processamento final no escalonamento. Isso ocorre desde que o tempo da tarefa de manutenção seja menor que o tempo de processamento adicional causado pela deterioração da máquina.

Na estratégia de manutenção 1, o objetivo é melhorar a solução obtida pelo algoritmo ILS adicionando as tarefas de manutenção (usando o Lema 1) após sua execução. Utilizando esta estratégia, o ILS pode realizar seus procedimentos (perturbação e busca local) utilizando todo o seu tempo de execução (τ). Em comparação com a estratégia sem manutenção, observa-se que esta estratégia realmente melhorou as soluções. Ao adicionar tarefas de manutenção às soluções exploradas durante a execução do algoritmo ILS/D, o tempo final de processamento pode ser consideravelmente reduzido.

Para avaliar os benefícios de adicionar tarefas de manutenção à vizinhança gerada pelo procedimento de busca local do algoritmo ILS, foi desenvolvida a estratégia de manutenção 2. O procedimento de perturbação e criação da vizinhança são realizados desconsiderando tarefas de manutenção para garantir que as soluções atendam aos requisitos do Lema 1 e outras condições (conforme descrito na Seção 4.6.2). Consequentemente, parte do tempo disponível para execução do ILS é utilizado para remover e adicionar tarefas de manutenção, reduzindo, assim, o número de soluções exploradas em um intervalo de tempo.

Com base nos experimentos realizados, descobriu-se que quando o algoritmo ILS tem um tempo de execução maior, ou seja, problemas de teste 100×5 ($\approx 1.6s$), 200×10 ($\approx 1.6s$) , e 200×5 ($\approx 3.2s$), a estratégia de manutenção 2 tem desempenho melhor que a estratégia 1. Por outro lado, nos problemas de teste com menor tempo de execução de ILS: 100×20 ($\approx 0.4s$), 100×10 ($\approx 0.8s$), e 200×20 ($\approx 0.8s$), a estratégia de manutenção 2 apresenta desempenho igual ou inferior à estratégia de manutenção 1. Em cenários com mais tempo disponível, adicionar e remover tarefas de manutenção pode levar a melhores soluções, de acordo com os resultados. No entanto, ao reduzir o tempo de execução do ILS, o custo computacional de adicionar e remover tarefas de manutenção pode resultar em uma diminuição na extensão do espaço de busca explorado.

A estratégia de manutenção 3 foi desenvolvida para avaliar os efeitos de tratar as tarefas de manutenção da mesma forma que qualquer outra tarefa no sequenciamento (escalonamento), ou seja, todas as operações realizadas pelos procedimentos de perturbação e busca local também afetam as tarefas de manutenção. Nesta estratégia, avaliou-se os efeitos da alocação de tarefas de manutenção do ILS/D sem levar em consideração o Lema 1 definido por Ruiz-Torres, Paletta e M'Hallah (2017). O Lema 1 é usado apenas para construir a população inicial.

De acordo com os resultados do algoritmo MS3, tratar a tarefa de manutenção como as outras tarefas do sequenciamento permite obter melhores resultados do que alocar a manutenção usando apenas o Lema 1, como foi realizado nos algoritmos MS1 e MS2. Na estratégia de manutenção 3 é possível explorar novas soluções dentro do espaço de busca, devido à alocação da tarefa de manutenção para uma posição determinada pelo algoritmo. A esse respeito, entende-se que determinar se deve ou não alocar a manutenção apenas com base na posição atual nem sempre resulta no melhor resultado (como ocorre no Lema 1). Assim, na estratégia de manutenção 3, ao realizar os procedimentos de busca local e perturbação, as inserções e trocas de tarefas (incluindo manutenção) permitem novas combinações de soluções. Embora o tempo de manutenção possa não ser o ideal naquele momento, ao escalonar a manutenção, pode-se reduzir o tempo de processamento final da máquina se considerar o tempo de processamento e a deterioração de todas as tarefas na máquina.

Com base nos resultados observados dos algoritmos MS1, MS2, MS3 e None, pode-se concluir que o Lema 1 é eficiente para escalonamento de manutenções. Isso fica claro quando os algoritmos MS1, MS2 e None são comparados. No entanto, entende-se que o Lema 1 restringe o algoritmo ILS/D de explorar soluções melhores, como demonstrado pela comparação de MS1, MS2 e MS3. Embora o algoritmo MS3 tenha sido estatisticamente inferior ao MS2, para os problemas de teste 200×5 na comparação par-a-par, quando consideram-se todas as instâncias (720), o algoritmo MS3 supera todos os outros algoritmos (Figura 15).

5.3.4 Validação dos Algoritmos

Finalmente, para validar os resultados dos algoritmos MS1, MS2 e MS3 para o problema abordado, esses algoritmos foram executados para 108 instâncias de pequeno porte, propostas neste trabalho. Essas instâncias usam combinações dos seguintes valores n e m: $n \in \{6, 7, 8\}$ e $m \in \{2, 3, 4\}$. Ao combinar esses parâmetros com os outros parâmetros (p_{jk}, d_{jk}, CR, u_k), explicados na Seção 5.3.1, 108 instâncias foram geradas no total.

Para encontrar os resultados exatos das instâncias de pequeno porte, foi utilizada a técnica de enumeração. Isso se deve à impossibilidade de executar o modelo devido à sua

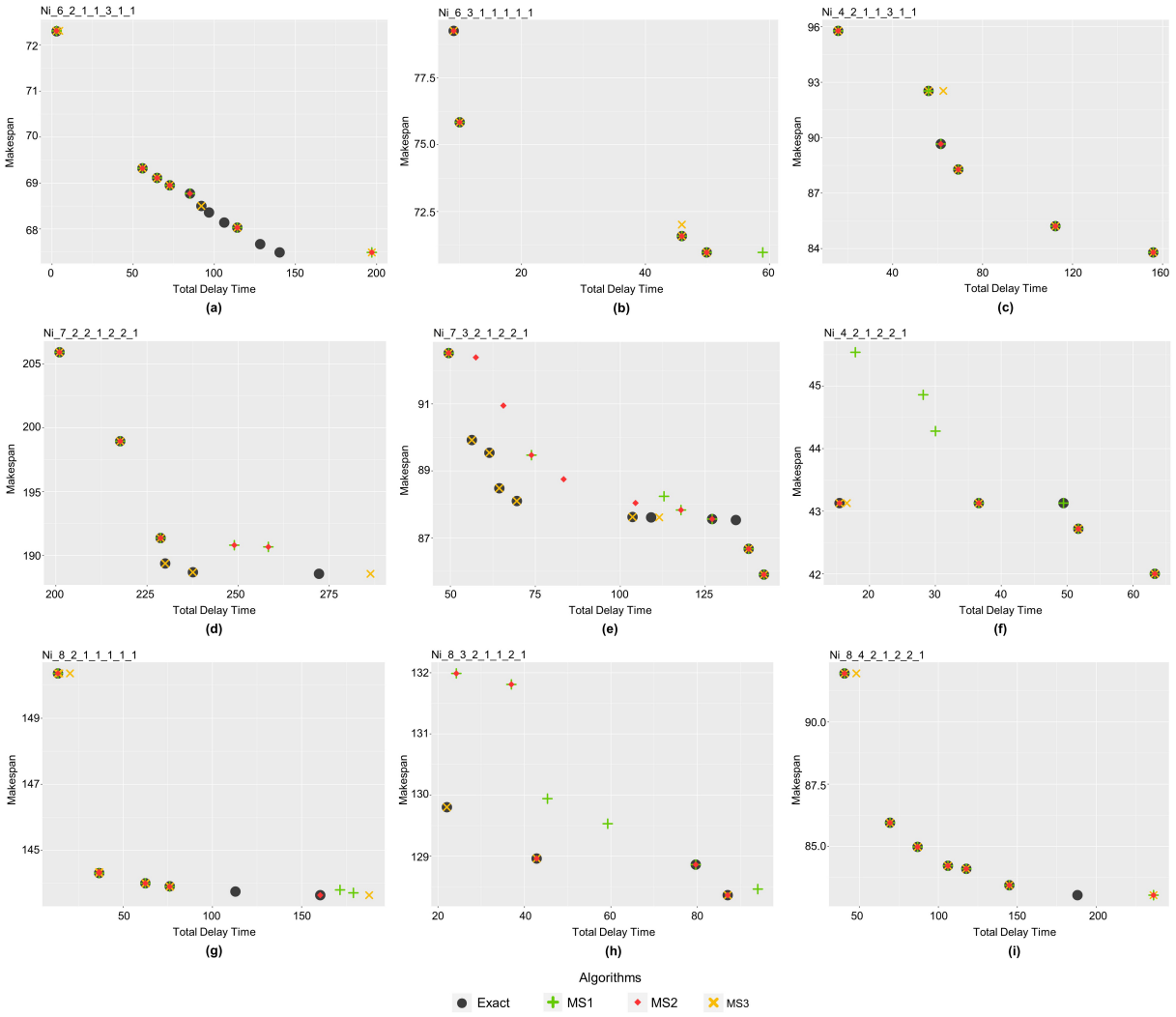


Figura 17 – Fronteira Pareto-ótima (círculos pretos) e as fronteiras retornadas pelos algoritmos MS1 (‘+’ verde), MS2 (retângulo vermelho) e MS3 (‘×’ amarelo).

não linearidade, como explicado na Seção 3.3.2. Assim, os resultados dos algoritmos são comparados com os valores ótimos obtidos pela técnica de enumeração. Contando todos os pontos exatos da fronteira Pareto-ótimo obtida pela técnica de enumeração, verificou-se que, em média, os algoritmos MS1, MS2 e MS3 encontram 77%, 85% e 87% pontos iguais, respectivamente .

A Figura 17 mostra os resultados obtidos pela técnica de enumeração e pelos algoritmos MS1, MS2 e MS3 para nove instâncias selecionadas aleatoriamente de cada combinação n e m. A partir da ilustração, pode-se ver que os resultados alcançados pelos algoritmos são semelhantes à técnica de enumeração.

Esses resultados confirmam a validade dos algoritmos MS1, MS2 e MS3 para o problema, mostrando que eles são capazes de convergir para soluções corretas da fronteira Pareto-ótimo.

5.4 Conclusão

Este capítulo apresentou a uma análise minuciosa e extensa dos experimentos computacionais realizados para avaliar a eficácia e o desempenho dos algoritmos desenvolvidos neste estudo. Esses experimentos foram conduzidos com o intuito de investigar o sequenciamento de produção em vários cenários, considerando a deterioração das máquinas e a integração das tarefas de manutenção.

Na primeira parte dos experimentos computacionais realizados, foram apresentadas as instâncias do problema utilizadas, destacando sua relevância e características específicas. Logo, os algoritmos foram avaliados em 900 instâncias de *benchmark* da literatura (RUIZ-TORRES; PALETTA; PEREZ-ROMAN, 2015). Em seguida, uma análise detalhada dos parâmetros críticos do algoritmo ILS/D foi realizada, demonstrando sua sensibilidade e a importância de ajustá-los adequadamente para alcançar soluções de alta qualidade.

Os resultados obtidos a partir da execução dos algoritmos em diferentes cenários e instâncias do problema foram apresentados e discutidos de maneira abrangente. Os resultados do algoritmo ILS/D foram comparados com os de outros algoritmos de referência, como MOEA/D, NSGA-II e PILS, para avaliar o desempenho relativo das abordagens multiobjetivo desenvolvidas neste trabalho.

Os resultados do experimento indicam claramente que a heurística proposta ILS/D ($g = 50$) é altamente eficaz em comparação com o clássico algoritmo MOEA/D ($g = 50$, $g = 500$, $g = 5000$ e $g = 50000$). Quando são comparados o algoritmo ILS/D e o algoritmo MOEA/D ($g = 500000$), o ILS/D apresenta resultados superiores apenas para os problemas de teste 100×5 e 200×5, que são considerados os problemas mais difíceis de resolver. Observou-se também que o ILS/D encontrou melhores resultados do que o algoritmo MOEA/D, na maioria dos casos para a métrica de distância. Além disso, verificou-se que o NSGA-II precisa de um grande número de gerações para superar o ILS/D, ou seja, o NSGA-II precisa de 50000 gerações. Adicionalmente, os resultados dos experimentos indicam que o ILS/D apresenta resultados superiores ao algoritmo PILS na maioria dos casos.

Na segunda parte dos experimentos computacionais realizados, as três estratégias para alocar a tarefa de manutenção no sequenciamento foram avaliadas. Os resultados demonstram manifestamente uma elevada capacidade do ILS/D para explorar estratégias de manutenção, demonstrando como essas estratégias podem afetar positivamente a qualidade das soluções obtidas. Além disso, os resultados experimentais comparativos também indicam que a estratégia de manutenção 3 é mais adequada para resolver o problema de escalonamento multiobjetivo em larga escala com deterioração da máquina e tarefas de manutenção integradas.

A principal conclusão que pode ser tirada deste estudo é que a abordagem de

decomposição e agregação pode ser uma maneira útil de estender o ILS aos problemas de otimização multiobjetivo. Além disso, pode-se concluir que algoritmos híbridos podem obter maior qualidade nas soluções em comparação com outras meta-heurísticas existentes na literatura.

Por fim, esses experimentos computacionais forneceram uma compreensão profunda do comportamento dos algoritmos desenvolvidos nos cenários testados e corroboraram a eficácia da abordagem multiobjetivo proposta.

6 Conclusões e Trabalhos Futuros

6.1 Considerações Finais

Neste capítulo final, são apresentadas as conclusões principais derivadas deste estudo abrangente sobre o sequenciamento de produção em cenários com efeito de deterioração das máquinas e integração das tarefas de manutenção. Os principais resultados e contribuições desta pesquisa são recapitulados, destacando suas implicações e relevância para a área de otimização de processos de fabricação.

6.2 Conclusões Principais

Nesta seção, as principais conclusões deste trabalho de pesquisa são resumidas.

Esta pesquisa estudou o problema de programação de máquinas paralelas não-relacionadas, estendendo-o para um problema biobjetivo que considera a minimização do tempo máximo de conclusão e minimização do tempo total de atraso das tarefas simultaneamente, quando o efeito de deterioração depende da sequência do processamento das tarefas nas máquinas. Adicionalmente, apresentou a integração das tarefas de manutenção no processo de sequenciamento, visando reduzir a deterioração das máquinas e, consequentemente, diminuir o tempo de processamento final. Este é um problema complexo e importante encontrado em uma ampla variedade de situações práticas.

É proposto um modelo de programação não-linear inteiro misto multiobjetivo para o problema. Uma vez que o problema é classificado como NP-difícil, é impraticável resolver instâncias de grande porte para otimizar de maneira exata. Sendo assim, para obter soluções próximas das ótimas, algoritmos heurísticos foram desenvolvidos.

Problemas de programação de produção com efeitos de deterioração têm sido amplamente estudados nos últimos anos. Em geral, os autores têm se concentrado no desenvolvimento de algoritmos mais rápidos para resolver o problema em sua versão mono-objetivo. O foco deste trabalho foi propor um algoritmo multiobjetivo para resolvê-lo.

O algoritmo ILS/D foi desenvolvido com sucesso, uma abordagem híbrida baseada em meta-heurísticas de população e busca local, que demonstrou excelente desempenho na resolução do problema multiobjetivo proposto. O ILS/D preservou a característica de diversificação da fronteira Pareto do MOEA/D e se mostrou eficaz em diferentes cenários.

Experimentos computacionais abrangentes foram realizados, comparando os resultados do algoritmo desenvolvido com outros algoritmos de referência. Os resultados do experimento indicam claramente que a heurística ILS/D proposta é altamente eficaz

quando comparada aos algoritmos MOEA/D, NSGA-II e PILS. A principal conclusão que pode ser tirada deste estudo é que a abordagem de decomposição e agregação pode ser uma maneira útil de estender o ILS aos problemas de otimização multiobjetivo. Além disso, pode-se concluir que algoritmos híbridos podem obter maior qualidade nas soluções em comparação com outras meta-heurísticas existentes na literatura.

Além disso, foram avaliadas três estratégias distintas para adicionar a tarefa de manutenção ao escalonamento. Os resultados mostraram como essas estratégias podem influenciar positivamente a qualidade das soluções obtidas, além de demonstrar uma elevada capacidade do ILS/D para explorar estratégias de manutenção no problema multiobjetivo com deterioração da máquina.

Essa análise experimental validou a eficácia das abordagens multiobjetivo propostas e proporcionou *insights* valiosos sobre o desempenho em diferentes cenários, reforçando a importância das estratégias desenvolvidas na otimização do sequenciamento de produção em ambientes industriais complexos.

6.3 Trabalhos Futuros

Nesta seção, possíveis direções para futuras pesquisas e desenvolvimentos na área são delineadas.

Como trabalhos futuros são propostas as seguintes atividades:

- Aprimorar o algoritmo incorporando técnicas mais eficientes para a geração de pesos e explorando outras funções de decomposição. Além disso, uma área promissora de investigação seria o desenvolvimento de métodos para gerar soluções iniciais de forma construtiva, como por exemplo, uma estratégia que ordene as tarefas começando pelas de maior tempo de processamento, visando obter um ponto de partida mais promissor para os algoritmos de otimização.

- Implementar novas buscas locais, explorando diferentes estruturas de vizinhança e investigando estratégias mais eficazes para evitar movimentos não promissores. Um exemplo concreto disso poderia ser o desenvolvimento de uma estrutura de vizinhança que permitisse a inserção única de tarefas, levando em consideração o tempo de atraso na entrega das tarefas como critério de decisão.

- Incluir no problema outras restrições que o tornem mais próximo de cenários realistas, como, por exemplo: associar o tempo de duração da manutenção à deterioração de cada máquina; estabelecer que cada máquina esteja vinculada a uma janela de tempo específica para a realização da manutenção preventiva, o que significa que a manutenção só pode ser iniciada dentro dessa janela determinada.

- Incorporar um objetivo adicional ao problema, como, por exemplo, a minimização do custo total de manutenção. Alternativamente, também poderia ser introduzido o custo total de manutenção como uma restrição adicional no problema. Além disso, seria interessante realizar uma análise mais aprofundada sobre o efeito do tempo de manutenção em cada máquina, investigando como essa variável pode impactar o sequenciamento e as decisões de manutenção.
- Explorar a adaptação das abordagens desenvolvidas para lidar com problemas de sequenciamento em tempo real, nos quais as informações estão em constante evolução, representa um desafio crucial a ser investigado.

6.4 Encerramento

Este estudo representa uma contribuição significativa para o campo da otimização de sequenciamento de produção. Espera-se que as abordagens desenvolvidas possam ser aplicadas para melhorar a eficiência e o desempenho dos sistemas de fabricação. Esta pesquisa abre novos caminhos para a investigação de problemas de sequenciamento multiobjetivo com efeito de deterioração e manutenção integrada, e almeja-se que inspire pesquisadores a explorar ainda mais essas áreas desafiadoras.

REFERÊNCIAS

ALFARES, H. K. Plant shutdown maintenance workforce team assignment and job scheduling. *Journal of Scheduling*, Springer, v. 25, n. 3, p. 321–338, 2022.

ALLAHVERDI, A. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, Elsevier, v. 246, n. 2, p. 345–378, 2015.

AQUINO, R. D.; CHAGAS, J. B. C.; SOUZA, M. J. F. Abordagem exata e heurísticas para o problema de planejamento de ordens de manutenção de longo prazo: Um estudo de caso industrial de larga escala. *Pesquisa Operacional para o Desenvolvimento*, v. 11, n. 3, p. 159–182, dez. 2019. Disponível em: ⟨https://revistapodes.emnuvens.com.br/podesenvolvimento/article/view/625⟩.

ARAÚJO, O. D.; DHEIN, G.; FAMPA, M. Minimizing the makespan on parallel machines with sequence dependent deteriorating effects. *XLIX Simpósio Brasileiro de Pesquisa Operacional. Blumenau-SC: SOBRAPO–Sociedade Brasileira Pesquisa Operacional*, p. 3556–67, 2017.

BIRATTARI, M. et al. A racing algorithm for configuring metaheuristics. In: *Gecco*. [S.l.: s.n.], 2002. v. 2, n. 2002.

BOSCHETTI, M. A. et al. Matheuristics: Optimization, simulation and control. In: SPRINGER. *International workshop on hybrid metaheuristics*. [S.l.], 2009. p. 171–177.

BROWNE, S.; YECHIALI, U. Scheduling deteriorating jobs on a single processor. *Operations Research*, INFORMS, v. 38, n. 3, p. 495–498, 1990.

BRUNO, J. L.; COFFMAN, E. G.; SETHI, R. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, ACM New York, NY, USA, v. 17, n. 7, p. 382–387, 1974.

CHASE, R.; JACOBS, F.; AQUILANO, N. *Administração da produção e operações para vantagens competitivas*. [S.l.]: McGraw-Hill, 2006. ISBN 9788586804694.

CHENG, W. et al. Variable neighborhood search for parallel machines scheduling problem with step deteriorating jobs. *Mathematical Problems in Engineering*, Hindawi Publishing Corporation, v. 2012, 2012.

CHRYSSOLOURIS, G. *Manufacturing systems: theory and practice*. [S.l.]: Springer Science & Business Media, 2013.

COELLO, C. A. C. *Evolutionary algorithms for solving multi-objective problems*. [S.l.]: Springer, 2007.

COTA, L. P. et al. An adaptive multi-objective algorithm based on decomposition and large neighborhood search for a green machine scheduling problem. *Swarm and Evolutionary Computation*, Elsevier, v. 51, p. 100601, 2019.

CROCE, F. D.; GARAIX, T.; GROSSO, A. Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem. *Computers & Operations Research*, Elsevier, v. 39, n. 6, p. 1213–1217, 2012.

DEB, K. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001. (Wiley Interscience Series in Systems and Optimization). ISBN 9780471873396. Disponível em: ⟨https://books.google.com.br/books?id=OSTn4GSy2uQC⟩.

DEB, K. Multi-objective optimisation using evolutionary algorithms: an introduction. In: *Multi-objective evolutionary optimisation for product design and manufacturing*. [S.l.]: Springer, 2011. p. 3–34.

DEB, K.; JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, IEEE, v. 18, n. 4, p. 577–601, 2013.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002.

DELORME, M.; IORI, M.; MENDES, N. F. Solution methods for scheduling problems with sequence-dependent deterioration and maintenance events. *European Journal of Operational Research*, Elsevier, 2021.

DING, J. et al. Parallel machine scheduling with completion-time-based criteria and sequence-dependent deterioration. *Computers & Operations Research*, Elsevier, v. 103, p. 35–45, 2019.

GAMRATH, G. et al. *The SCIP Optimization Suite 7.0*. [S.l.], 2020. Disponível em: ⟨http://www.optimization-online.org/DB_HTML/2020/03/7705.html⟩.

GARA-ALI, A.; ESPINOUSE, M.-L.; FINKE, G. Unrelated parallel-machine scheduling with deterioration effects and multi-maintenance activities. In: *CIE-44-44th International Conference on Computers & Industrial Engineering*. [S.l.: s.n.], 2014.

GAREY, M. R.; JOHNSON, D. S. Computers and intractability. a guide to the theory of np-completeness. WH Freeman and Company, 1979.

GAWIEJNOWICZ, S. A review of four decades of time-dependent scheduling: Main results, new topics, and open problems. *Journal of Scheduling*, Springer, v. 23, n. 1, p. 3–47, 2020.

GEIGER, M. J. The pils metaheuristic and its application to multi-objective machine scheduling. *Multicriteria Decision Making and Fuzzy Systems Theory, Methods and Applications*, p. 43–58, 2006.

GEIGER, M. J. Foundations of the pareto iterated local search metaheuristic. *arXiv preprint arXiv:0809.0406*, 2008.

GEIGER, M. J. Improvements for multi-objective flow shop scheduling by pareto iterated local search. *arXiv preprint arXiv:0907.2993*, 2009.

GEIGER, M. J. Decision support for multi-objective flow shop scheduling by the pareto iterated local search methodology. *Computers & industrial engineering*, Elsevier, v. 61, n. 3, p. 805–812, 2011.

GLOVER, F. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, v. 65, n. 1, p. 223–253, 1996. ISSN 0166-218X. First International Colloquium on Graphs and Optimization. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/0166218X9400037E⟩.

GOMES, H. C.; NEVES, F. d. A. das; SOUZA, M. J. F. Multi-objective metaheuristic algorithms for the resource-constrained project scheduling problem with precedence relations. *Computers & Operations Research*, Elsevier, v. 44, p. 92–104, 2014.

GRAHAM, R. L. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, Elsevier, v. 5, p. 287–326, 1979.

GUPTA, J. N.; GUPTA, S. K. Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, v. 14, n. 4, p. 387–393, jan. 1988. ISSN 03608352.

HANSEN, P.; MLADENOVIĆ, N.; PEREZ, J. A. M. Variable neighbourhood search: methods and applications. *4OR*, Springer, v. 6, p. 319–360, 2008.

HOFFMAN, M. et al. Online Maintenance Prioritization Via Monte Carlo Tree Search and Case-Based Reasoning. *Journal of Computing and Information Science in Engineering*, v. 22, n. 4, p. 041005, 02 2022. ISSN 1530-9827. Disponível em: ⟨https://doi.org/10.1115/1.4053408⟩.

HSU, C.-J. et al. Unrelated parallel-machine scheduling problems with aging effects and deteriorating maintenance activities. *Information Sciences*, Elsevier, v. 253, p. 163–169, 2013.

HSU, C.-J.; YANG, D.-L. Unrelated parallel-machine scheduling with position-dependent deteriorating jobs and resource-dependent processing time. *Optimization Letters*, Springer, v. 8, n. 2, p. 519–531, 2014.

HUANG, C.; LI, Y.; YAO, X. A survey of automatic parameter tuning methods for metaheuristics. *IEEE transactions on evolutionary computation*, IEEE, v. 24, n. 2, p. 201–216, 2019.

HUANG, X.; WANG, M.-Z. Parallel identical machines scheduling with deteriorating jobs and total absolute differences penalties. *Applied Mathematical Modelling*, Elsevier, v. 35, n. 3, p. 1349–1353, 2011.

HUANG, X.; WANG, M.-Z.; JI, P. Parallel machines scheduling with deteriorating and learning effects. *Optimization Letters*, Springer, v. 8, n. 2, p. 493–500, 2014.

JACOB, V. V. *Aplicação de metaheurísticas para problemas de sequenciamento com lotes de tarefas*. Dissertação (Mestrado) — Universidade Federal de Viçosa, https://locus.ufv.br//handle/123456789/2669, 4-Jul 2014.

JI, M.; CHENG, T. E. Parallel-machine scheduling of simple linear deteriorating jobs. *Theoretical Computer Science*, Elsevier, v. 410, n. 38, p. 3761–3768, 2009.

JIANG, S. Lagrangian relaxation for parallel machine batch scheduling with deteriorating jobs. In: IEEE. *Management of e-Commerce and e-Government (ICMeCG), 2011 Fifth International Conference on*. [S.l.], 2011. p. 109–112.

JOO, C.; KIM, B. Machine scheduling of time-dependent deteriorating jobs with determining the optimal number of rate modifying activities and the position of the activities. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, v. 9, n. 1, p. JAMDSM0007–JAMDSM0007, 2015.

KAGERMANN, H. et al. *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group*. [S.l.]: Forschungsunion, 2013.

KRUSKAL, W. H.; WALLIS, W. A. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, Taylor & Francis Group, v. 47, n. 260, p. 583–621, 1952.

KUNNATHUR, A.; GUPTA, S. Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research*, v. 47, p. 56–64, 1990.

LALLA-RUIZ, E.; VOß, S. Modeling the parallel machine scheduling problem with step deteriorating jobs. *European Journal of Operational Research*, v. 255, n. 1, p. 21–33, 2016. ISSN 0377-2217. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0377221716302296⟩.

LEE, H.-T.; YANG, D.-L.; YANG, S.-J. Multi-machine scheduling with deterioration effects and maintenance activities for minimizing the total earliness and tardiness costs. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 66, n. 1-4, p. 547–554, 2013.

LEE, W.-C. et al. A single-machine scheduling problem with two-agent and deteriorating jobs. *Applied Mathematical Modelling*, Elsevier Inc., v. 34, n. 10, p. 3098–3107, out. 2010. ISSN 0307904X.

LEVITIN, G.; XING, L.; DAI, Y. Optimal operation and maintenance scheduling in m-out-of-n standby systems with reusable elements. *Reliability Engineering System Safety*, v. 211, p. 107582, 2021. ISSN 0951-8320. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0951832021001320⟩.

LI, H.; ZHANG, Q. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 13, n. 2, p. 284–302, 2008.

LIU, W. et al. A maintenance activity scheduling with time-and-position dependent deteriorating effects. *Math. Biosci. Eng*, v. 19, p. 11756–11767, 2022.

LÓPEZ-IBÁÑEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, Elsevier, v. 3, p. 43–58, 2016.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 320–353.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: *Handbook of Metaheuristics*. [S.l.]: Springer, 2010. p. 363–397.

LUO, Y. Stochastic single machine scheduling with time-dependent deterioration or position-dependent learning effect. *Optimization Letters*, Springer, p. 1–22, 2022.

LUSTOSA, L.; MESQUITA, M. A.; OLIVEIRA, R. J. *Planejamento e controle da produção*. [S.l.]: Elsevier Brasil, 2008.

MA, W.-M. et al. Parallel-machine scheduling with delivery times and deteriorating maintenance. *Asia-Pacific Journal of Operational Research*, World Scientific, 2015.

MAZDEH, M. M. et al. Parallel machines scheduling to minimize job tardiness and machine deteriorating cost with deteriorating jobs. *Applied Mathematical Modelling*, v. 34, n. 6, p. 1498 – 1510, 2010. ISSN 0307-904X. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0307904X09002789⟩.

MIETTINEN, K. *Nonlinear multiobjective optimization*. [S.l.]: Springer Science & Business Media, 1999. v. 12.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & Operations Research*, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

MONTOYA-TORRES, J. R. et al. Modeling the parallel machine scheduling problem with worker- and position-dependent processing times. In: DOLGUI, A. et al. (Ed.). *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems*. Cham: Springer International Publishing, 2021. p. 351–359. ISBN 978-3-030-85906-0.

MOSHEIOV, G. V-shaped policies for scheduling deteriorating jobs. *Operations Research*, INFORMS, v. 39, n. 6, p. 979–991, 1991.

MOSHEIOV, G. Scheduling jobs under simple linear deterioration. *Computers & Operations Research*, Elsevier, v. 21, n. 6, p. 653–659, 1994.

MOSHEIOV, G. λ-shaped policies to schedule deteriorating jobs. *Journal of the Operational Research Society*, Taylor & Francis, v. 47, n. 9, p. 1184–1191, 1996.

MOSHEIOV, G. A note on scheduling deteriorating jobs. *Mathematical and Computer Modelling*, v. 41, n. 8, p. 883–886, 2005. ISSN 0895-7177. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0895717705001500⟩.

MOSHEIOV, G. A note: Multi-machine scheduling with general position-based deterioration to minimize total load. *International Journal of Production Economics*, Elsevier, v. 135, n. 1, p. 523–525, 2012.

PAQUETE, L.; CHIARANDINI, M.; STÜTZLE, T. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: *Metaheuristics for multiobjective optimisation*. [S.l.]: Springer, 2004. p. 177–199.

PEI, J. et al. Parallel-machine serial-batching scheduling with release times under the effects of position-dependent learning and time-dependent deterioration. *Annals of Operations Research*, Springer, v. 298, n. 1-2, p. 407–444, 2021.

PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, Springer, v. 19, n. 2, p. 201–232, 2013.

PÉREZ, E.; AMBATI, R. R.; RUIZ-TORRES, A. J. Maximising the number of on-time jobs on parallel servers with sequence dependent deteriorating processing times and periodic maintenance. *International Journal of Operational Research*, Inderscience Publishers (IEL), v. 32, n. 3, p. 267–289, 2018.

PINEDO, M. L. *Scheduling*. [S.l.]: Springer, 2012. v. 29.

PINEDO, M. L. *Scheduling: Theory, Algorithms, and Systems*. 5th. ed. [S.l.]: Springer International Publishing, 2016. ISBN 978-3-319-26580-3.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. *International Journal of Production Research*, Taylor & Francis, v. 51, n. 17, p. 5238–5252, 2013.

RUIZ-TORRES, A.; PALETTA, G.; PéREZ, E. Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research*, 2013.

RUIZ-TORRES, A. J.; ABLANEDO-ROSAS, J. H.; MELNIK, D. J. Flowshop scheduling with machine deterioration based on job sequences. *Ingeniería y competitividad: revista científica y tecnológica*, Universidad del Valle, v. 23, n. 2, p. 5, 2021.

RUIZ-TORRES, A. J.; PALETTA, G.; M'HALLAH, R. Makespan minimisation with sequence-dependent machine deterioration and maintenance events. *International Journal of Production Research*, Taylor & Francis, v. 55, n. 2, p. 462–479, 2017.

RUIZ-TORRES, A. J.; PALETTA, G.; PEREZ-ROMAN, E. Maximizing the percentage of on-time jobs with sequence dependent deteriorating process times. *International Journal of Operations Research and Information Systems (IJORIS)*, IGI Global, v. 6, n. 3, p. 1–18, 2015.

RUSSOMANO, V. *PCP, planejamento e controle da produção*. [S.l.]: Pioneira, 2000. (Biblioteca Pioneira de administração e negocios). ISBN 9788522100088.

SADEGHEIH, A. Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on ga performance. *Applied Mathematical Modelling*, Elsevier, v. 30, n. 2, p. 147–154, 2006.

SÁNCHEZ-HERRERA, S.; MONTOYA-TORRES, J. R.; SOLANO-CHARRIS, E. L. Flow shop scheduling problem with position-dependent processing times. *Computers Operations Research*, v. 111, p. 325–345, 2019. ISSN 0305-0548. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0305054819301753⟩.

SANTOS, V. L. A. *Sequenciamento de tarefas em máquinas paralelas com desgastes dependentes da sequência: resolução heurística*. Dissertação (Mestrado) — Universidade Federal de Viçosa, https://locus.ufv.br//handle/123456789/9402, 6-Jul 2016.

SANTOS, V. L. A.; ARROYO, J. E. C. Iterated greedy with random variable neighborhood descent for scheduling jobs on parallel machines with deterioration effect. *Electronic Notes in Discrete Mathematics*, Elsevier, v. 58, p. 55–62, 2017.

SANTOS, V. L. A.; ARROYO, J. E. C.; CARVALHO, T. F. Iterated local search based heuristic for scheduling jobs on unrelated parallel machines with machine deterioration effect. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. [S.l.: s.n.], 2016. p. 53–54.

SANTOS, V. L. A. et al. Multi-objective iterated local search based on decomposition for job scheduling problems with machine deterioration effect. *Engineering Applications of Artificial Intelligence*, v. 112, p. 104826, 2022. ISSN 0952-1976. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0952197622000926⟩.

SANTOS, V. L. A. et al. Multi-objective meta-heuristics for a scheduling problem with sequence-dependent machine deterioration. *Anais do LIV Simpósio Brasileiro de Pesquisa Operacional; Juiz de Fora-MG. BR. Campinas : Galoá; 2022*, v. 54, p. 152781, 2022. Disponível em: ⟨https://proceedings.science/sbpo/sbpo-2022/trabalhos/multi-objective-meta-heuristics-for-a-scheduling-problem-with-sequence-dependent?lang=pt-br⟩.

SCHEFFÉ, H. Experiments with mixtures. *Journal of the Royal Statistical Society: Series B (Methodological)*, Wiley Online Library, v. 20, n. 2, p. 344–360, 1958.

SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, JSTOR, v. 52, n. 3/4, p. 591–611, 1965.

SIMÕES, E. M. L.; BATISTA, L. D. S.; SOUZA, M. J. F. A matheuristic algorithm for the multiple-depot vehicle and crew scheduling problem. *IEEE Access*, v. 9, p. 155897–155923, 2021.

SOUZA, M. J. F. et al. A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operations Research*, Elsevier, v. 207, n. 2, p. 1041–1051, 2010.

SRINIVAS, N.; DEB, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, MIT Press, v. 2, n. 3, p. 221–248, 1994.

STÜTZLE, T. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, Elsevier, v. 174, n. 3, p. 1519–1539, 2006.

SUBRAMANIAN, A. et al. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, v. 37, n. 11, p. 1899 – 1911, 2010.

TOKSARI, M. D.; GÜNER, E. Parallel machine earliness/tardiness scheduling problem under the effects of position based learning and linear/nonlinear deterioration. *Computers & Operations Research*, Elsevier, v. 36, n. 8, p. 2394–2417, 2009.

TRIVEDI, A. et al. A survey of multiobjective evolutionary algorithms based on decomposition. *IEEE Transactions on Evolutionary Computation*, v. 21, n. 3, p. 440–462, 2017.

VARGAS, D. et al. Um algoritmo baseado em evolução diferencial para problemas de otimização estrutural multiobjetivo com restrições. *Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería*, Elsevier, v. 32, n. 2, p. 91–99, 2016.

WANG, J.-B.; WEI, C.-M. Parallel machine scheduling with a deteriorating maintenance activity and total absolute differences penalties. *Applied Mathematics and Computation*, Elsevier, v. 217, n. 20, p. 8093–8099, 2011.

WANG, J.-J.; WANG, J.-B.; LIU, F. Parallel machines scheduling with a deteriorating maintenance activity. *Journal of the Operational Research Society*, Nature Publishing Group, v. 62, n. 10, p. 1898–1902, 2011.

XU, X. Q.; LEI, D. M. A parallel iterated local search for two-agent flow shop scheduling. In: TRANS TECH PUBL. *Advanced Materials Research*. [S.l.], 2014. v. 926, p. 3476–3484.

YANG, D.-L. et al. Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities. *Computers Operations Research*, v. 39, n. 7, p. 1458–1464, 2012. ISSN 0305-0548. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0305054811002498⟩.

YANG, S.-J. Parallel machines scheduling with simultaneous considerations of position-dependent deterioration effects and maintenance activities. *Journal of the Chinese Institute of Industrial Engineers*, Taylor Francis, v. 28, n. 4, p. 270–280, 2011. Disponível em: ⟨https://doi.org/10.1080/10170669.2011.573006⟩.

ZHANG, Q.; LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, IEEE, v. 11, n. 6, p. 712–731, 2007.

ZHENG, Q.-Q. et al. An effective hybrid meta-heuristic for flexible flow shop scheduling with limited buffers and step-deteriorating jobs. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 106, p. 104503, 2021.

ZHOU, A. et al. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and evolutionary computation*, Elsevier, v. 1, n. 1, p. 32–49, 2011.

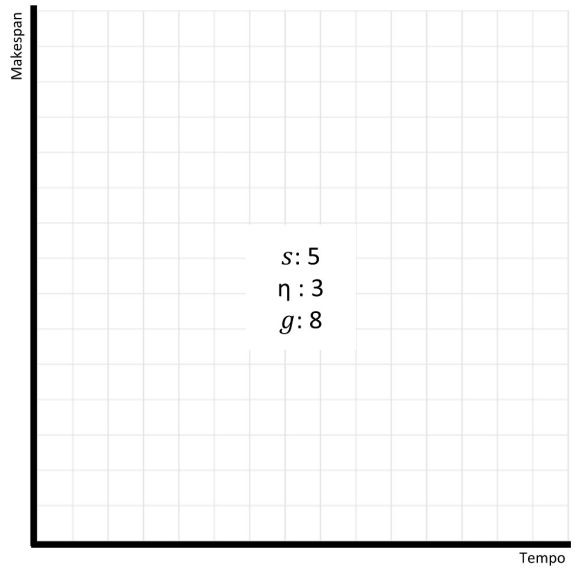
ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, IEEE, v. 3, n. 4, p. 257–271, 1999.

A Ilustração do Funcionamento Algoritmo ILS/D

As imagens a seguir fornecem uma representação visual das etapas executadas pelo ILS/D em um cenário hipotético. Estas ilustrações relacionam as linhas do pseudocódigo do algoritmo com um plano cartesiano, que representa o espaço de dois objetivos. Essa abordagem visual facilita a compreensão das operações realizadas pelo ILS/D durante a busca por soluções na fronteira Pareto estimada.

Em resumo, essas imagens (em formato de slides) proporcionam uma visão clara e intuitiva das operações do ILS/D no contexto do problema biobjetivo, ajudando a compreender como o algoritmo navega pelo espaço de objetivos em busca de soluções eficazes.

Os parâmetros de entrada do ILS/D são:
s: total de subproblemas gerados por decomposição;
η : número de vizinhos de cada subproblema;
g: número máximo de gerações.
Os parâmetros τ, t_{min} e t_{max} são utilizados pelo ILS.

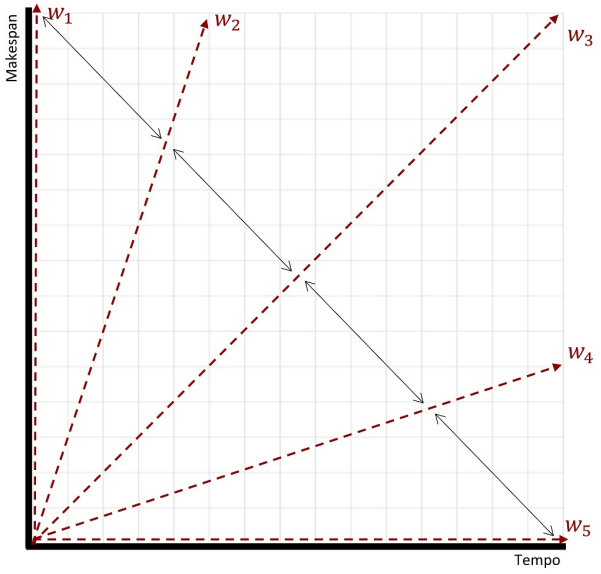


Algoritmo 4: ILS/D

entrada: total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

Um conjunto de pesos, denominados W, é construído pelo método *Scheffé* para o número s de subproblemas.

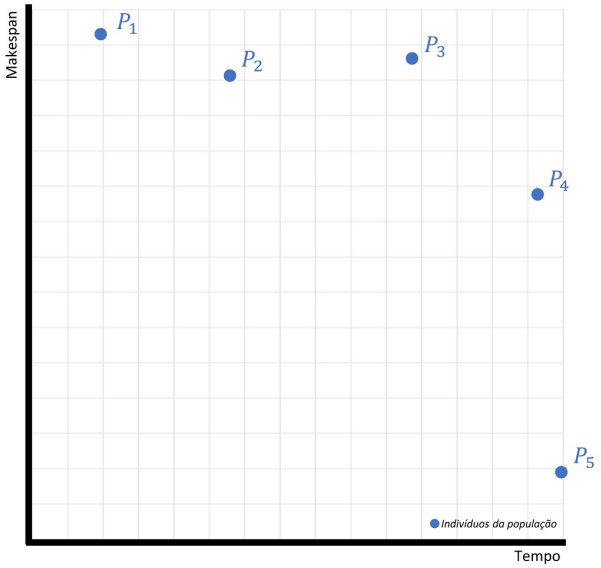


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

P é construído com s indivíduos gerados pelo procedimento construtivo.

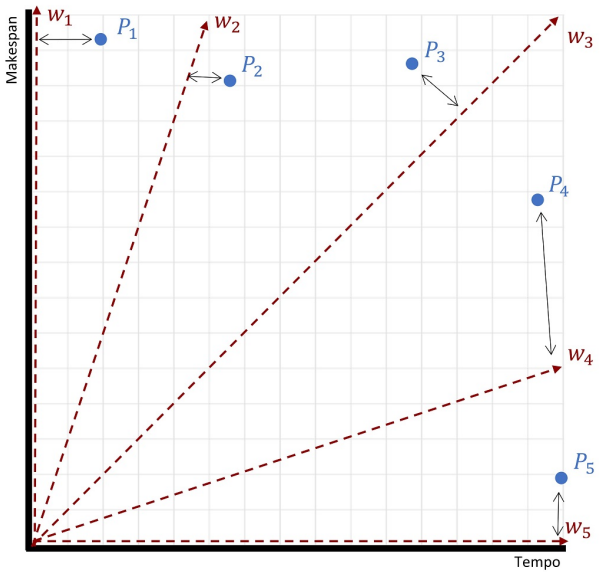


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

Cada indivíduo na população (P_i) representa uma solução candidata para o subproblema correspondente (i) e está associado a um vetor de peso (w_i).

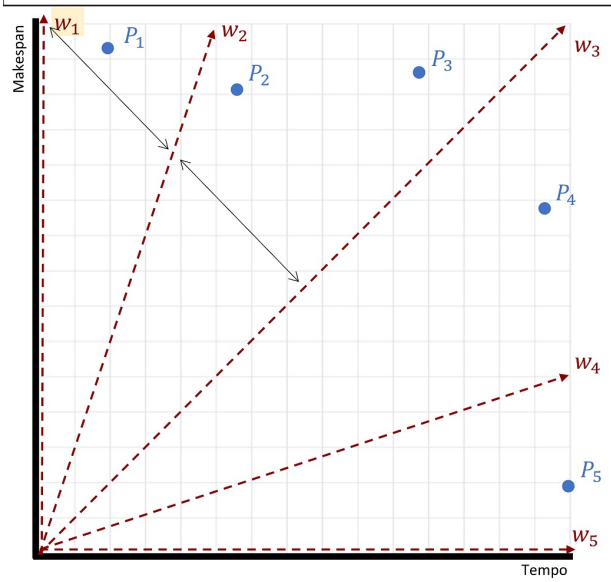


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

O conjunto de η vizinhos de cada subproblema i é definido para B_i, que contém η vetores de pesos mais próximos de w_i, $v \in B_i$.
Para definir esses subproblemas vizinhos, é calculada a distância Euclidiana entre w_i de P_i e todos os outros pesos do conjunto W.
Assim, aqueles η vizinhos que têm o menor valor de distância são definidos como o conjunto B_i.

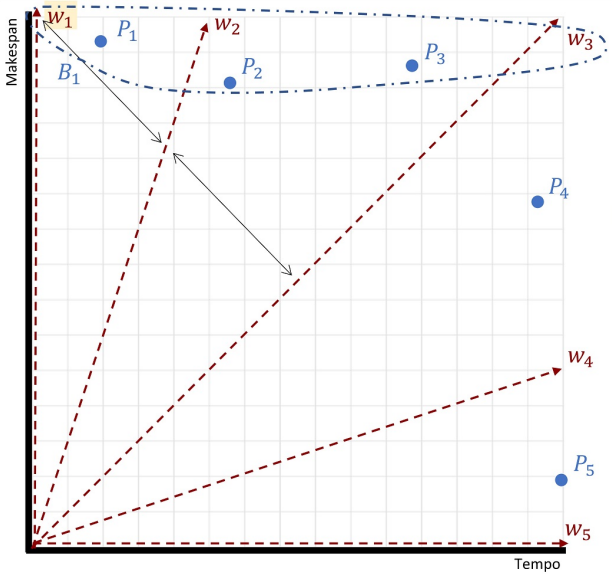


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

O conjunto de η vizinhos de cada subproblema i é definido para B_i, que contém η vetores de pesos mais próximos de $w_i, v \in B_i$.
Para definir esses subproblemas vizinhos, é calculada a distância Euclidiana entre w_i de P_i e todos os outros pesos do conjunto W.
Assim, aqueles η vizinhos que têm o menor valor de distância são definidos como o conjunto B_i.

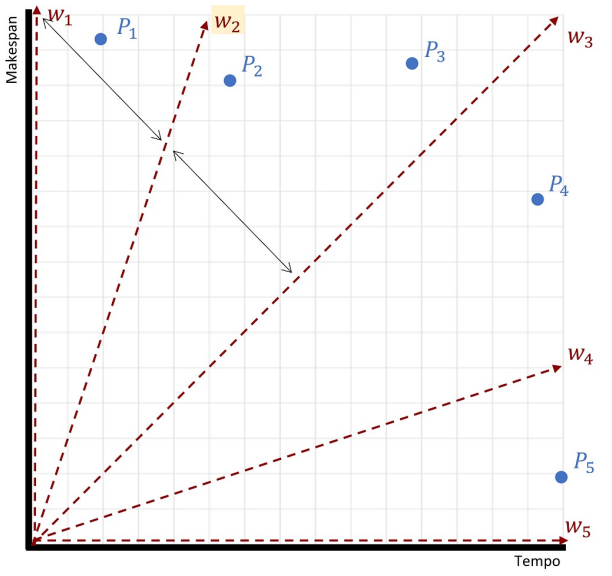


Algoritmo 4: ILS/D

entrada: total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

O conjunto de η vizinhos de cada subproblema i é definido para B_i, que contém η vetores de pesos mais próximos de $w_i, v \in B_i$.
Para definir esses subproblemas vizinhos, é calculada a distância Euclidiana entre w_i de P_i e todos os outros pesos do conjunto W.
Assim, aqueles η vizinhos que têm o menor valor de distância são definidos como o conjunto B_i.

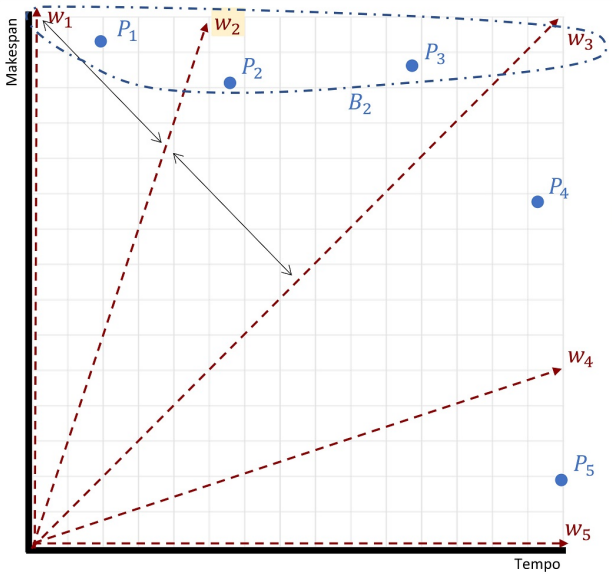


Algoritmo 4: ILS/D

entrada: total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

O conjunto de η vizinhos de cada subproblema i é definido para B_i, que contém η vetores de pesos mais próximos de $w_i, v \in B_i$.
Para definir esses subproblemas vizinhos, é calculada a distância Euclidiana entre w_i de P_i e todos os outros pesos do conjunto W.
Assim, aqueles η vizinhos que têm o menor valor de distância são definidos como o conjunto B_i.

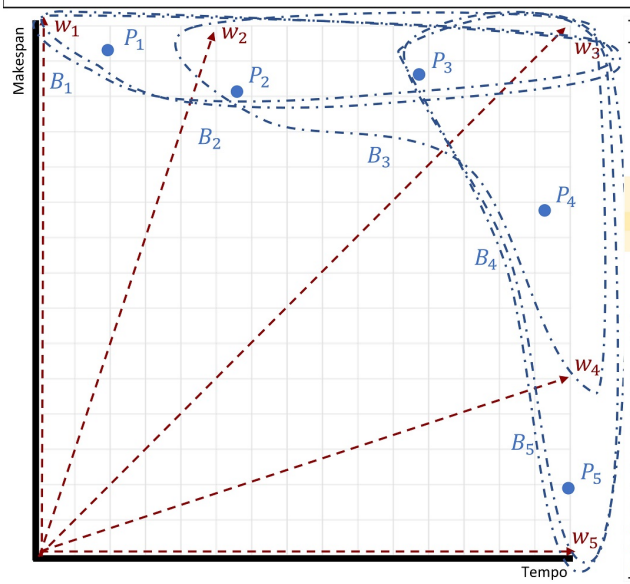


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

$W \leftarrow$ gere s vetores de peso usando o método Scheffé;
$P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
para $i = 1$ *to* s **faça**
  Associe P_i com w_i;
  $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
fim
$z^* \leftarrow \emptyset$;
Atualize z^* com o melhor valor de cada objetivo;
para $y = 1$ **to** g **faça**
  para $i = 1$ **to** s **faça**
    Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
    $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
    Atualize z^* considerando os valores dos objetivos de P_i;
    para *cada vizinho* $w_j \in B_i$ **faça**
      se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
        $P_j \leftarrow P_i$;
      fim
    fim
  fim
fim
Remova as soluções dominadas em P;
retorna P;

O conjunto de η vizinhos de cada subproblema i é definido para B_i, que contém η vetores de pesos mais próximos de $w_i, v \in B_i$.
Para definir esses subproblemas vizinhos, é calculada a distância Euclidiana entre w_i de P_i e todos os outros pesos do conjunto W.
Assim, aqueles η vizinhos que têm o menor valor de distância são definidos como o conjunto B_i.

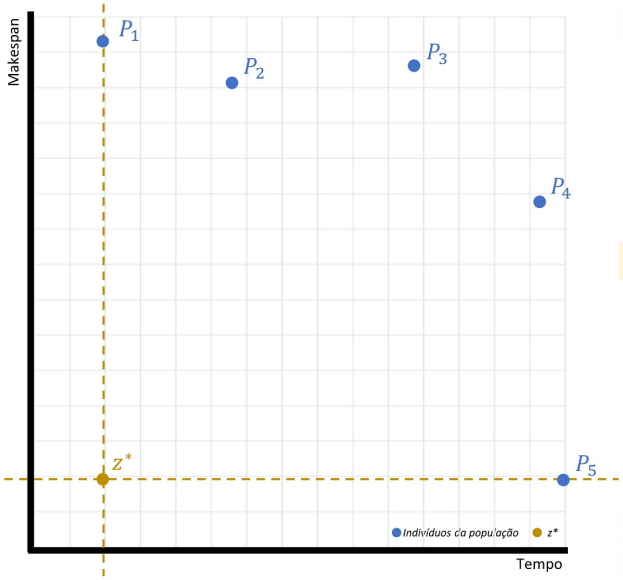


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

$W \leftarrow$ gere s vetores de peso usando o método Scheffé;
$P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
para $i = 1$ *to* s **faça**
  Associe P_i com w_i;
  $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
fim
$z^* \leftarrow \emptyset$;
Atualize z^* com o melhor valor de cada objetivo;
para $y = 1$ **to** g **faça**
  para $i = 1$ **to** s **faça**
    Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
    $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
    Atualize z^* considerando os valores dos objetivos de P_i;
    para *cada vizinho* $w_j \in B_i$ **faça**
      se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
        $P_j \leftarrow P_i$;
      fim
    fim
  fim
fim
Remova as soluções dominadas em P;
retorna P;

A estimativa do ponto ideal no espaço objetivo é denotada como z^*.
Este é definido pelo ponto formado pelo menor valor de *makespan* e o menor valor de TDT encontrado durante a execução do algoritmo até o momento, ou seja, z^* é atualizado com a melhor solução para cada objetivo.
z^* é inicializado e atualizado.

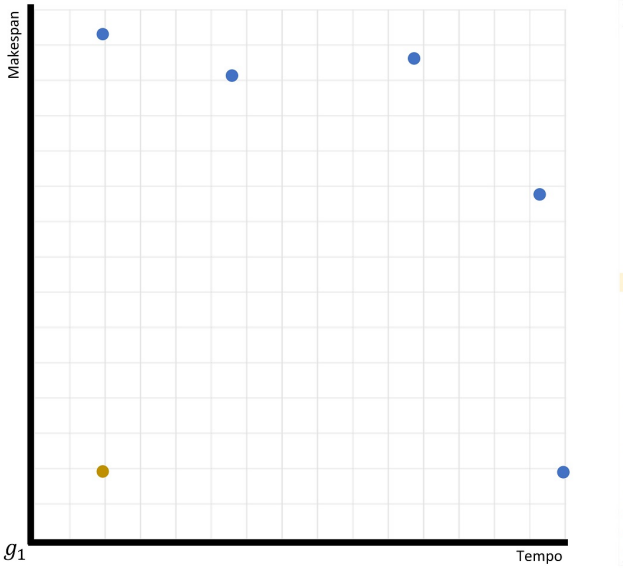


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

O laço mais externo que controla cada geração, até um máximo de g gerações é definido.

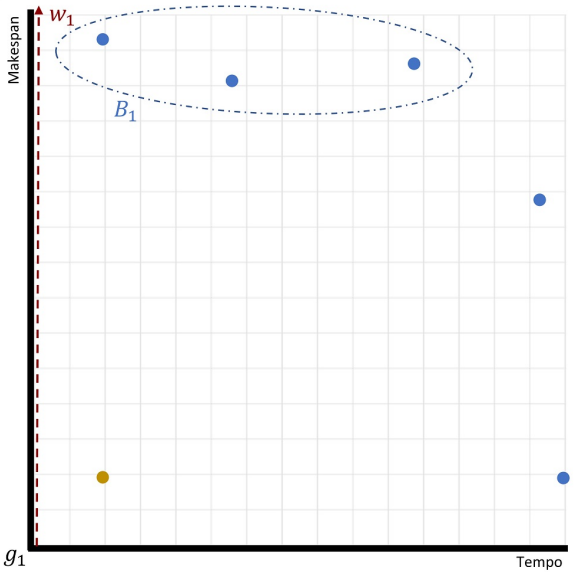


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

É definido o laço interno que controla cada um dos subproblemas i até o número total s .

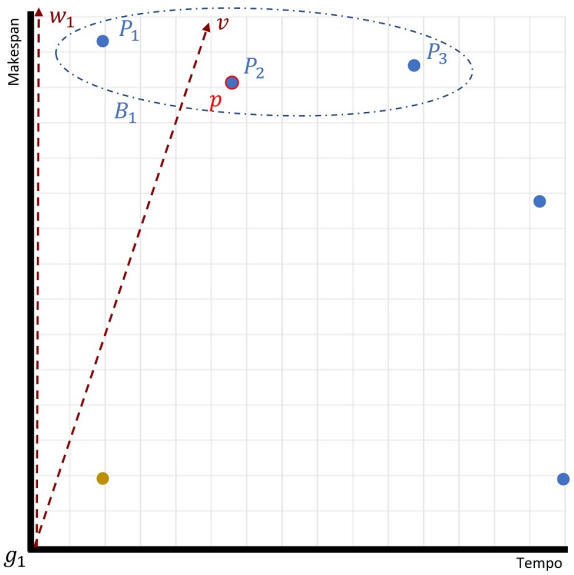


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |    associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

O vizinho v do subproblema i é selecionado aleatoriamente a partir de B_i.
A solução correspondente associada a v é passada como parâmetro ao algoritmo ILS, tornando-se a solução inicial.

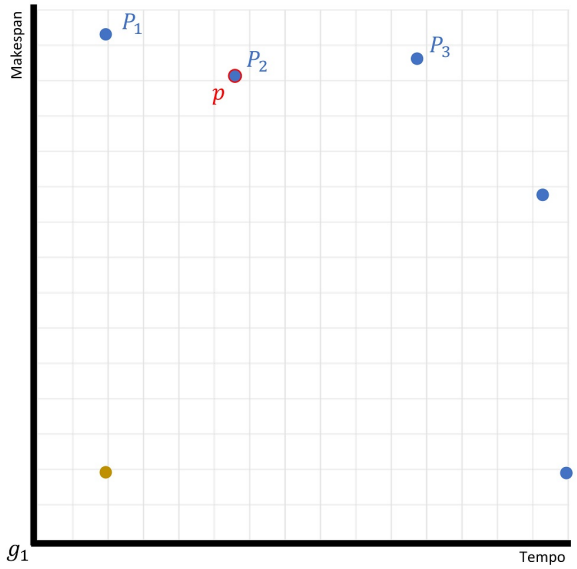


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |    associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

A solução correspondente associada a v é passada como parâmetro ao algoritmo ILS, tornando-se a solução inicial.
P_i é atualizado com a solução otimizada por ILS.

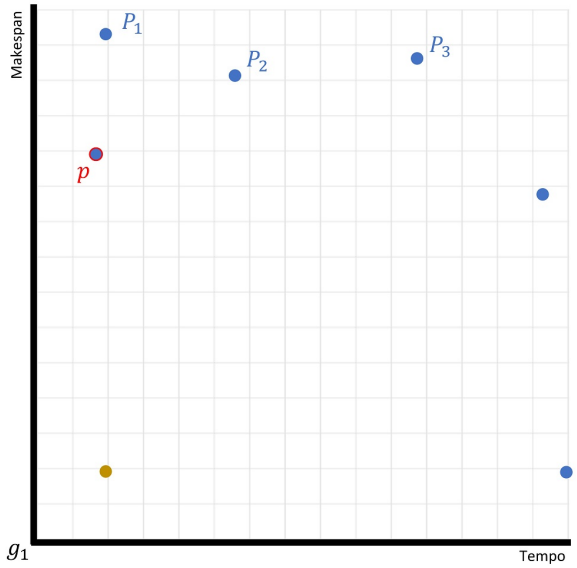


Algoritmo 4: ILS/D

entrada: total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

A solução correspondente associada a v é passada como parâmetro ao algoritmo ILS, tornando-se a solução inicial.
P_i é atualizado com a solução otimizada por ILS.

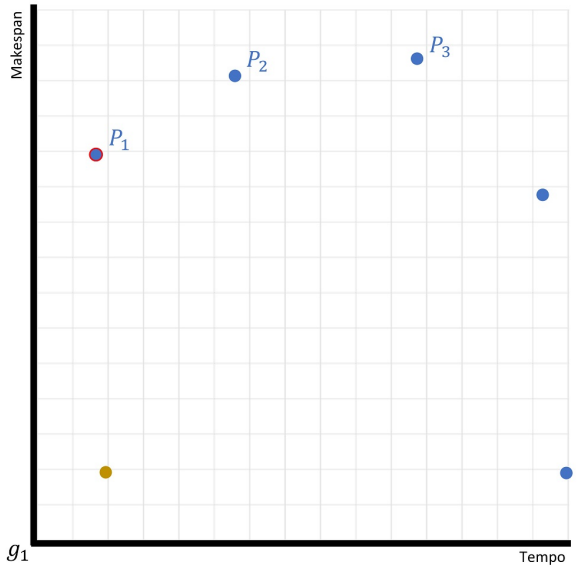


Algoritmo 4: ILS/D

entrada: total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

A solução correspondente associada a v é passada como parâmetro ao algoritmo ILS, tornando-se a solução inicial.
P_i é atualizado com a solução otimizada por ILS.

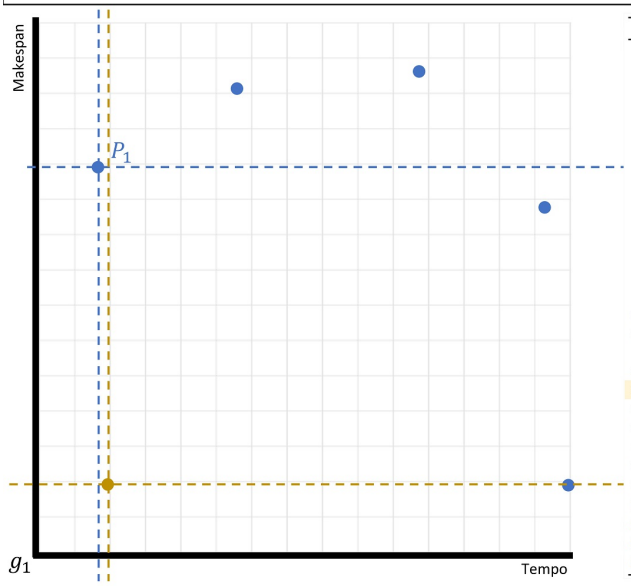


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

O vetor das melhores soluções z^* é atualizado.

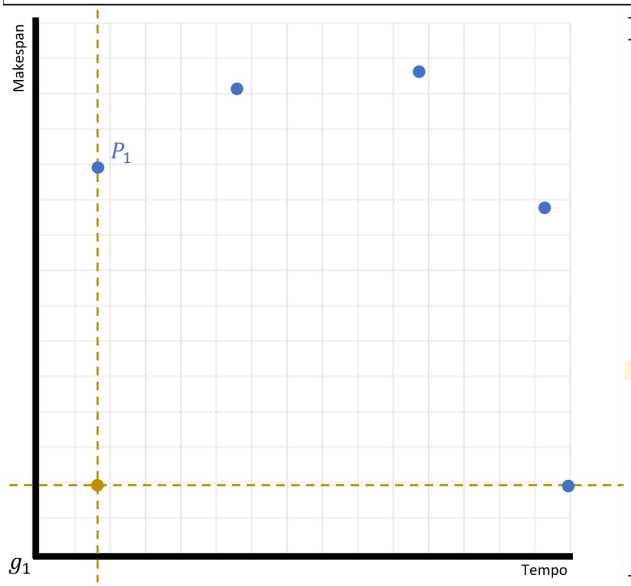


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

O vetor das melhores soluções z^* é atualizado.

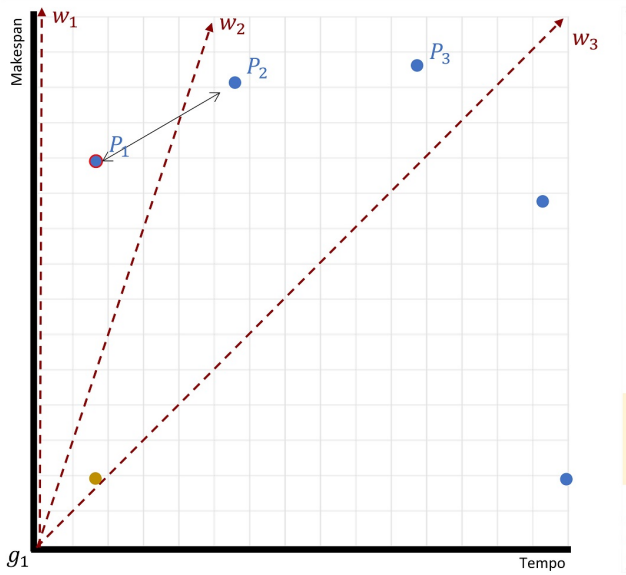


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

Usando a função de agregação *Tchebycheff*, verifica se a solução P_i é melhor do que a solução P_j, para cada vizinho $w_j \in B_i$.
Se verdadeiro, P_j é atualizado com a solução P_i.
O número máximo de atualizações que podem ser feitas é definido por $\frac{\eta}{2}$.

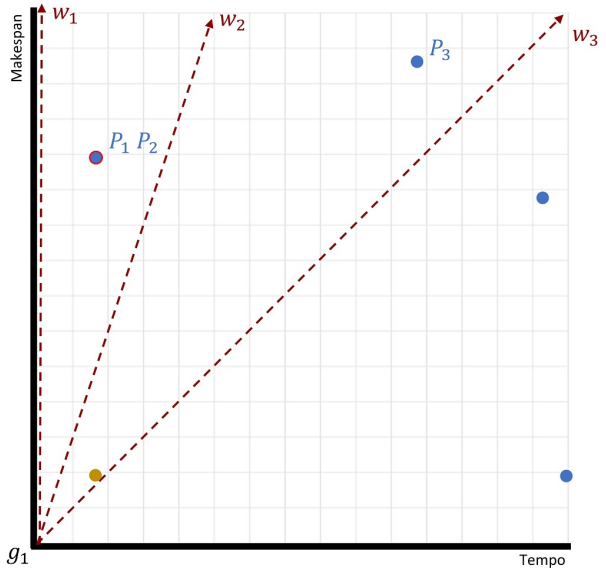


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ **to** g **faça**
10 | **para** $i = 1$ **to** s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

Usando a função de agregação *Tchebycheff*, verifica se a solução P_i é melhor do que a solução P_j, para cada vizinho $w_j \in B_i$.
Se verdadeiro, P_j é atualizado com a solução P_i.
O número máximo de atualizações que podem ser feitas é definido por $\frac{\eta}{2}$.

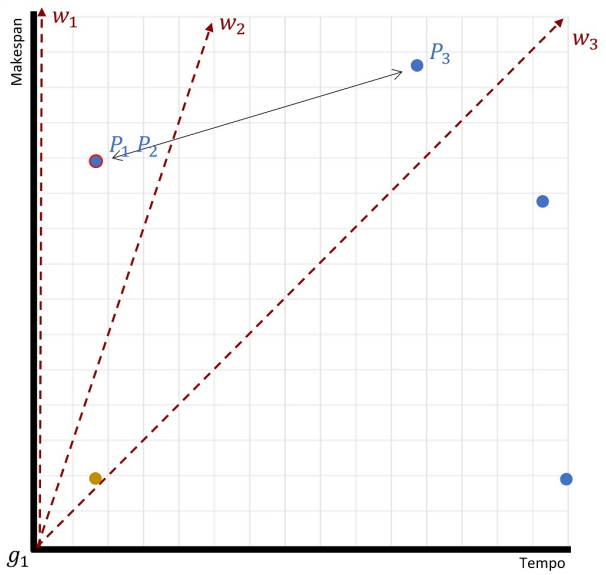


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ to g **faça**
10 | **para** $i = 1$ to s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

Usando a função de agregação *Tchebycheff*, verifica se a solução P_i é melhor do que a solução P_j, para cada vizinho $w_j \in B_i$.
Se verdadeiro, P_j é atualizado com a solução P_i.
O número máximo de atualizações que podem ser feitas é definido por $\frac{\eta}{2}$.

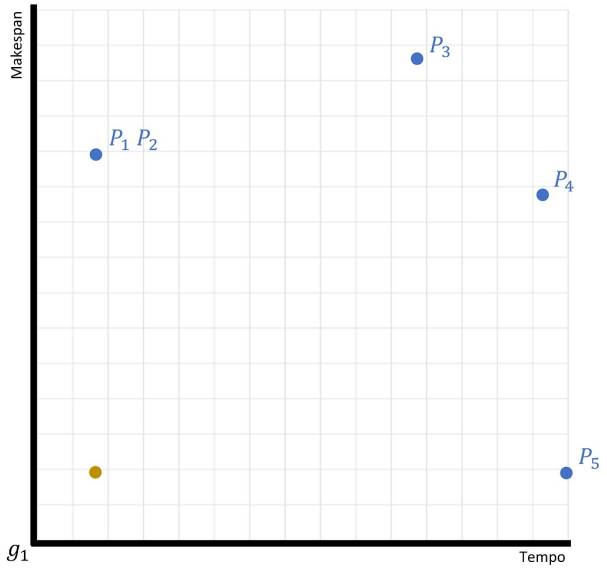


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ to g **faça**
10 | **para** $i = 1$ to s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow \text{ILS}(\tau, t_{min}, t_{max}, p, w_i, z^*)$;
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

Finaliza a primeira iteração do laço mais interno.

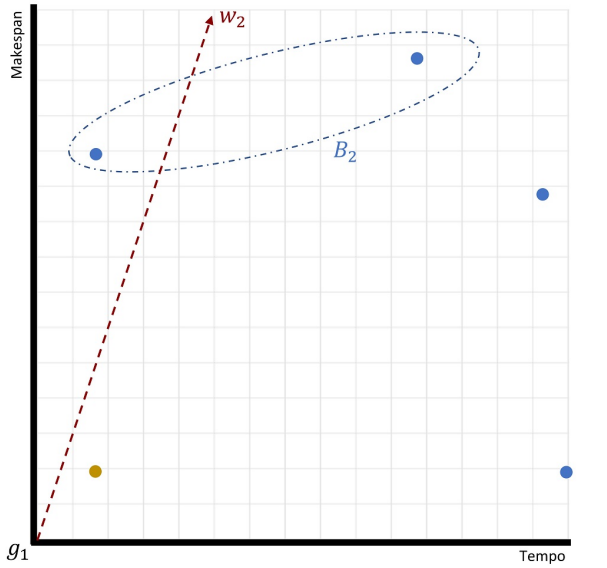


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

O vizinho v do subproblema i é selecionado aleatoriamente a partir de B_i.

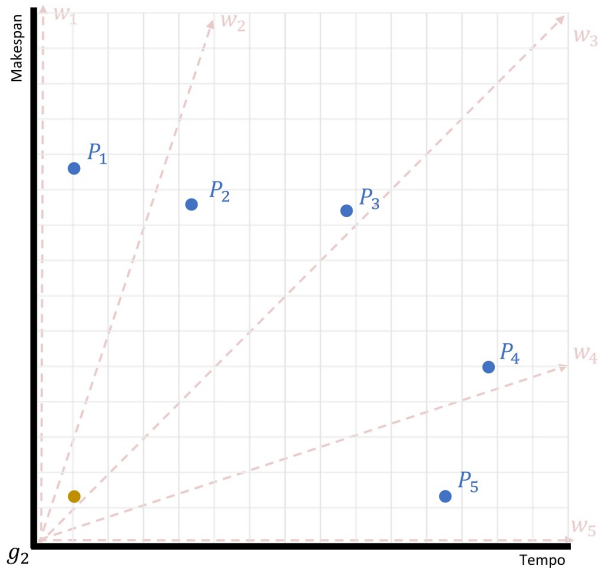


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

Realiza a segunda iteração do laço mais externo.

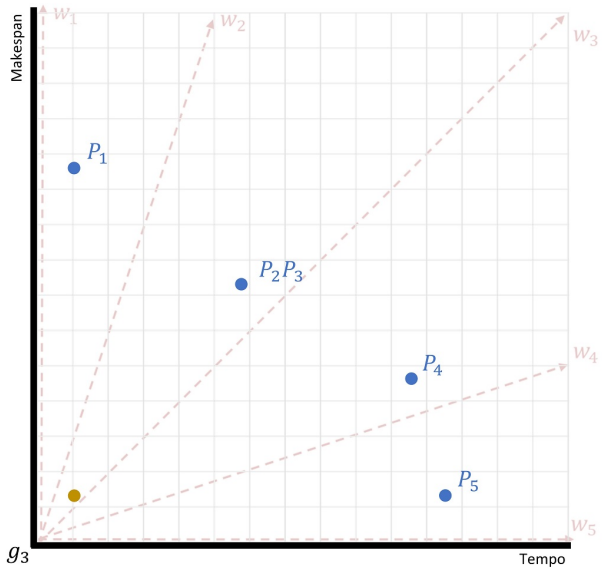


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

Realiza a terceira iteração do laço mais externo.

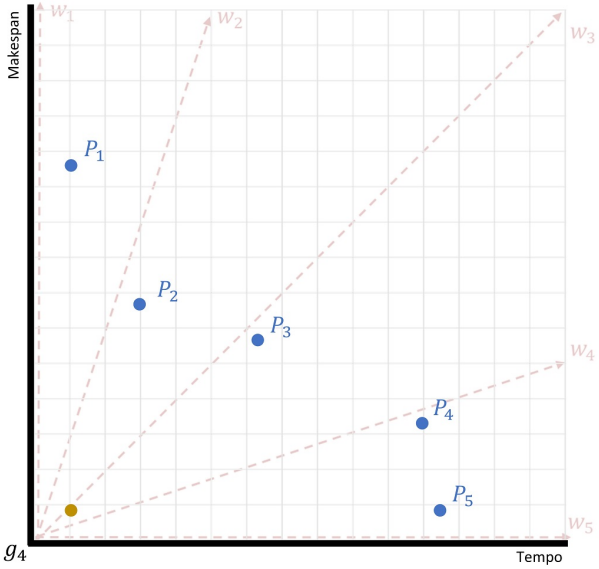


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

Realiza a quarta iteração do laço mais externo.

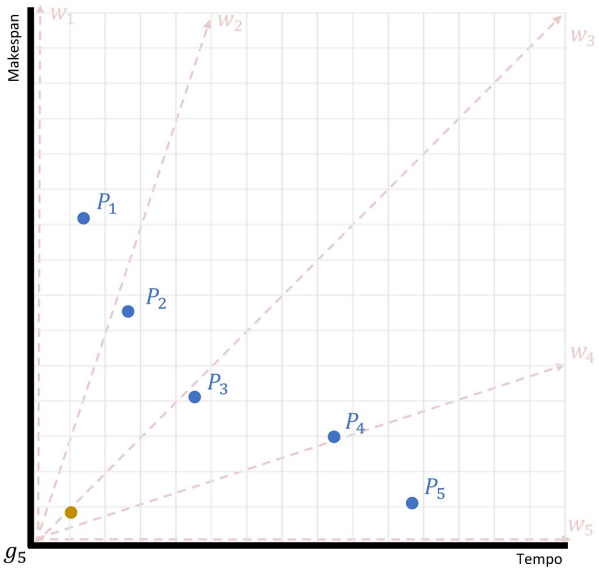


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

Realiza a quinta iteração do laço mais externo.

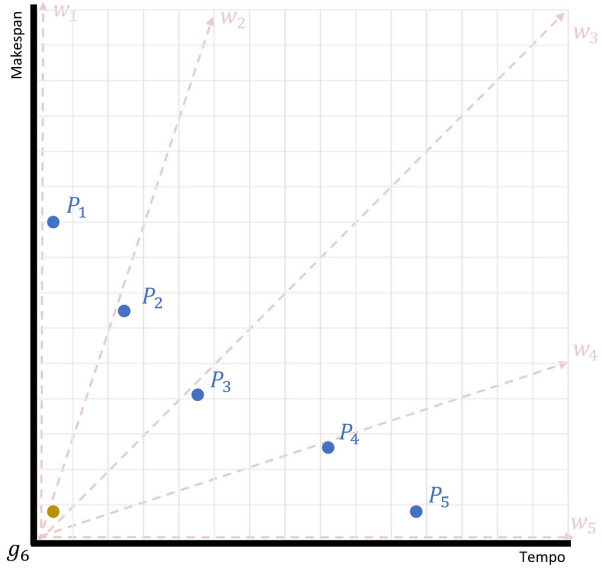


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

Realiza a sexta iteração do laço mais externo.

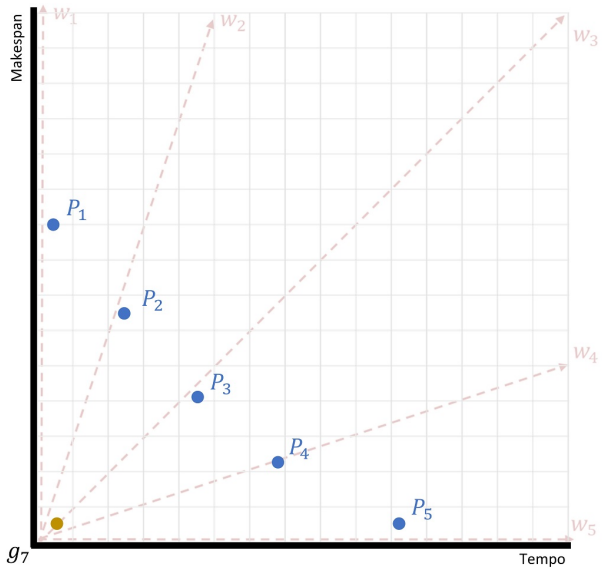


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ to g **faça**
10 | **para** $i = 1$ to s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow$ ILS(τ, t_{min}, t_{max}, p, w_i, z^*);
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

Realiza a sétima iteração do laço mais externo.

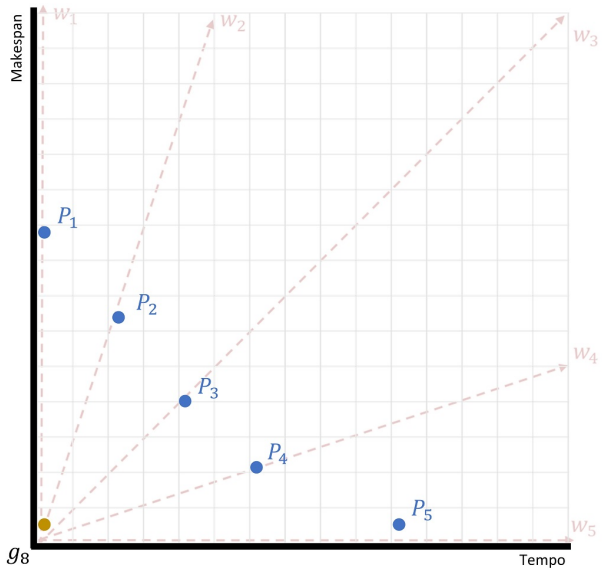


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

1 $W \leftarrow$ gere s vetores de peso usando o método Scheffé;
2 $P \leftarrow$ gere s soluções iniciais com o procedimento construtivo;
3 **para** $i = 1$ *to* s **faça**
4 | Associe P_i com w_i;
5 | $B_i \leftarrow \{v \in W \mid v$ é um dos η vizinhos mais próximos de $w_i\}$;
6 **fim**
7 $z^* \leftarrow \emptyset$;
8 Atualize z^* com o melhor valor de cada objetivo;
9 **para** $y = 1$ to g **faça**
10 | **para** $i = 1$ to s **faça**
11 | | Selecione aleatoriamente um vizinho v não visitado de B_i com solução associada $p \in P$;
12 | | $P_i \leftarrow$ ILS(τ, t_{min}, t_{max}, p, w_i, z^*);
13 | | Atualize z^* considerando os valores dos objetivos de P_i;
14 | | **para** *cada vizinho* $w_j \in B_i$ **faça**
15 | | | se $f^{te}(P_i|w_j, z^*) \leq f^{te}(P_j|w_j, z^*)$ **então**
16 | | | | $P_j \leftarrow P_i$;
17 | | | **fim**
18 | | **fim**
19 | **fim**
20 **fim**
21 Remova as soluções dominadas em P;
22 **retorna** P;

Realiza a oitava iteração do laço mais externo.

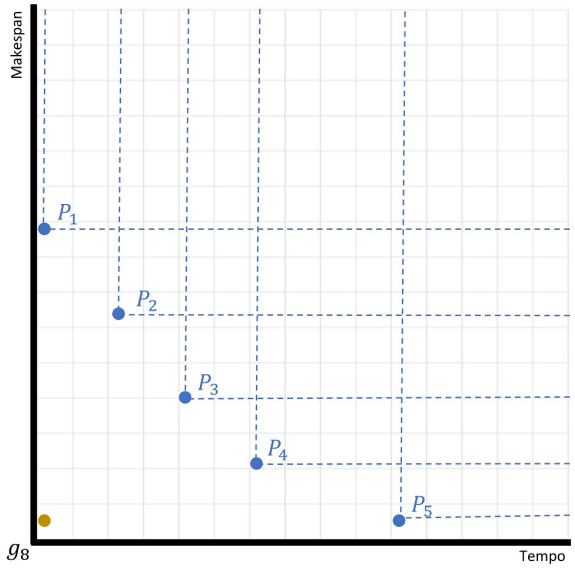


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

As soluções que são dominadas na população P são removidas.

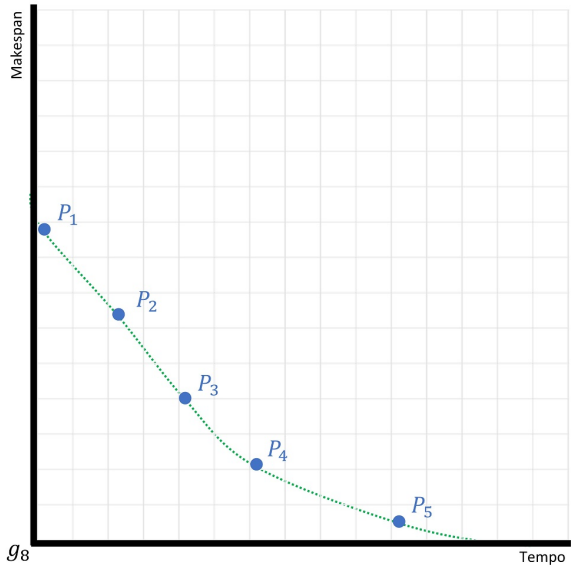


Algoritmo 4: ILS/D

entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

O conjunto P restante é retornado.



Algoritmo 4: ILS/D

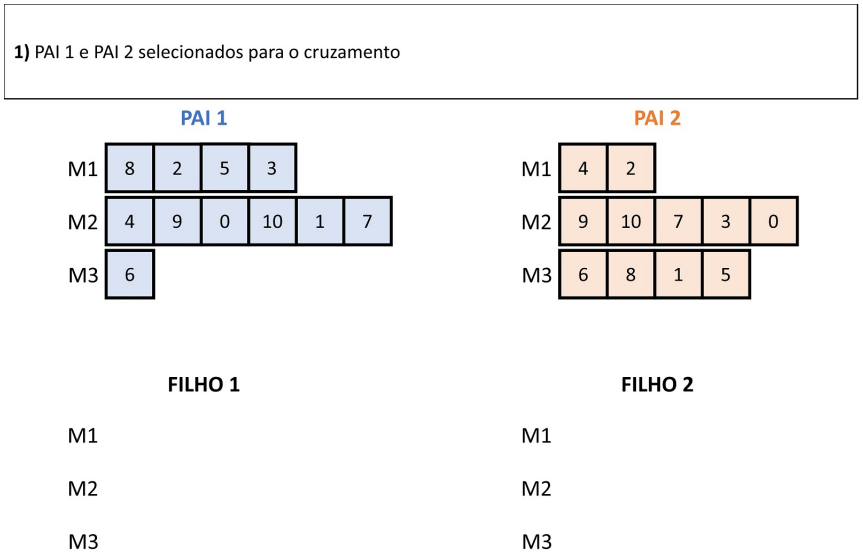
entrada : total de subproblemas s, número de vizinhos de cada subproblema η, número máximo de gerações g, tempo máximo de execução do ILS τ, tamanho mínimo de perturbação t_{min}, tamanho máximo de perturbação t_{max}
saída : população P

```
1  W ← gere s vetores de peso usando o método Scheffé;
2  P ← gere s soluções iniciais com o procedimento construtivo;
3  para i = 1 to s faça
4  |  Associe P_i com w_i;
5  |  B_i ← {v ∈ W | v é um dos η vizinhos mais próximos de w_i};
6  fim
7  z* ← ∅;
8  Atualize z* com o melhor valor de cada objetivo;
9  para y = 1 to g faça
10 |  para i = 1 to s faça
11 |  |  Selecione aleatoriamente um vizinho v não visitado de B_i com solução
   |  |   associada p ∈ P;
12 |  |  P_i ← ILS(τ, t_min, t_max, p, w_i, z*);
13 |  |  Atualize z* considerando os valores dos objetivos de P_i;
14 |  |  para cada vizinho w_j ∈ B_i faça
15 |  |  |  se f^te(P_i|w_j, z*) ≤ f^te(P_j|w_j, z*) então
16 |  |  |  |  P_j ← P_i;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 Remova as soluções dominadas em P;
22 retorna P;
```

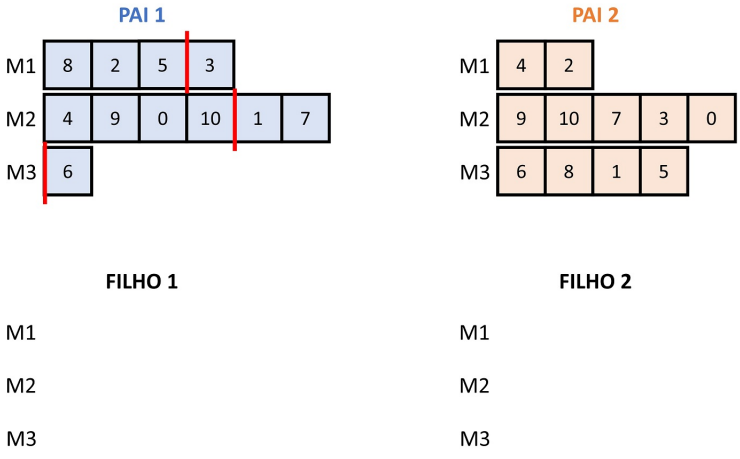
B Ilustração das Operações de Cruzamento e Mutação

As imagens a seguir oferecem uma representação visual das operações de cruzamento e mutação que são empregadas nos algoritmos NSGA-II e MOEA/D. O operador de cruzamento foi parcialmente baseado no operador de cruzamento para sequenciamento de uma única máquina apresentado por Sadegheih (2006). Quanto ao operador de mutação, foram selecionados aleatoriamente seis exemplos de movimentos a partir das estruturas de vizinhanças, apresentadas na Seção 4.5.2.2, e aplicados na prole. Essas imagens em formato de slides têm como objetivo tornar as operações mais compreensíveis.

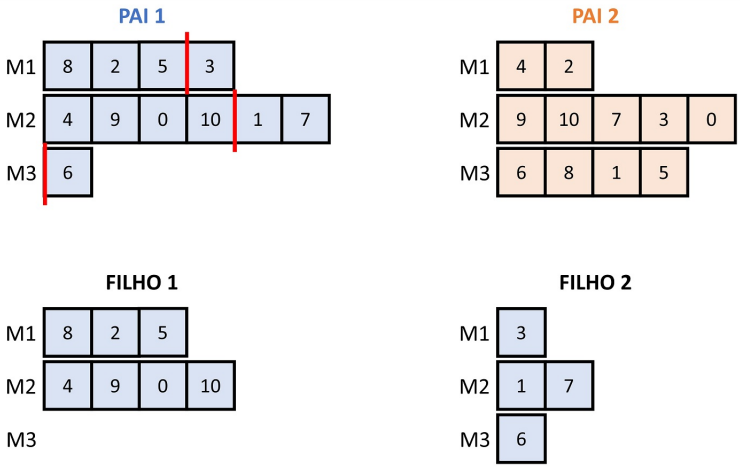
Em resumo, essa série de imagens proporciona uma visão clara e intuitiva das operações de cruzamento e mutação propostas neste trabalho para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não-Relacionadas com a Deterioração da Máquina Dependente da Sequência. Elas facilitam a compreensão do funcionamento dessas operações que foram implementadas.



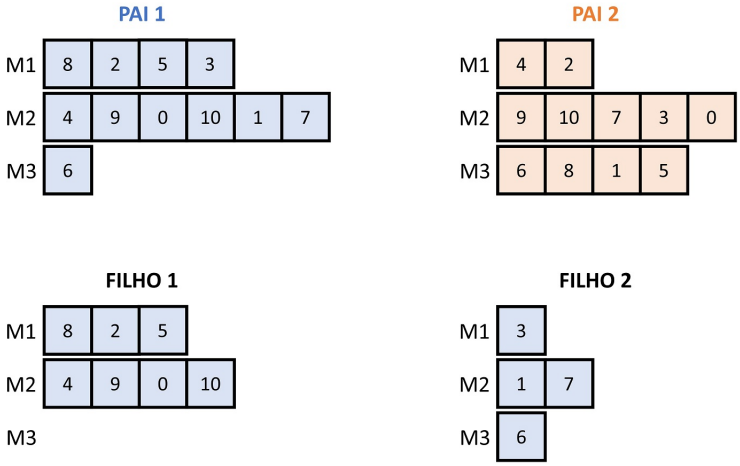
2) Adiciona as tarefas do PAI 1 nos filhos
- Para cada máquina é definido um ponto de corte aleatório



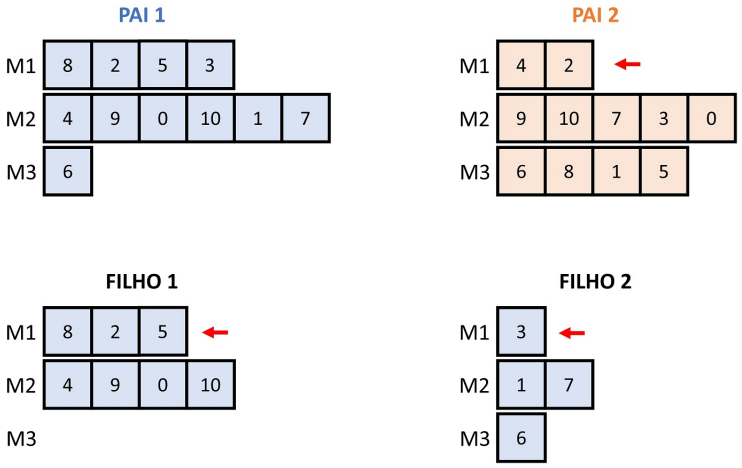
2) Adiciona as tarefas do PAI 1 nos filhos
- Para cada máquina é definido um ponto de corte aleatório
- As tarefas são distribuídas entre os filhos



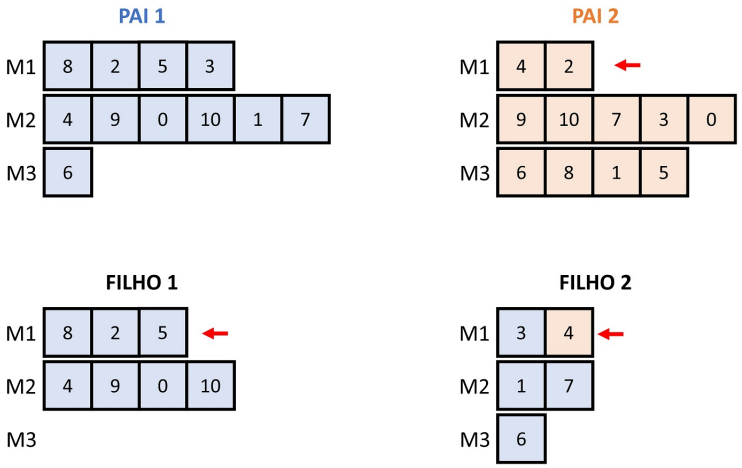
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



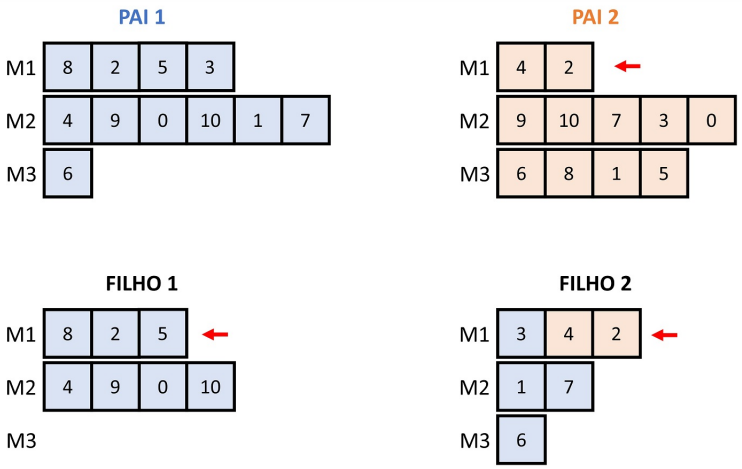
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



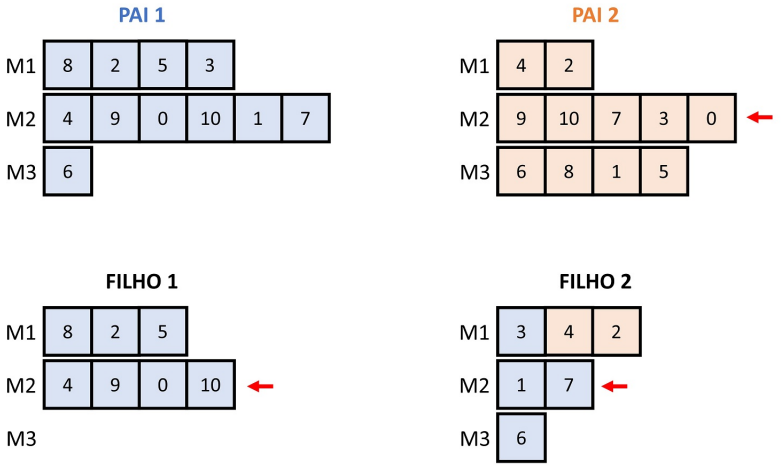
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



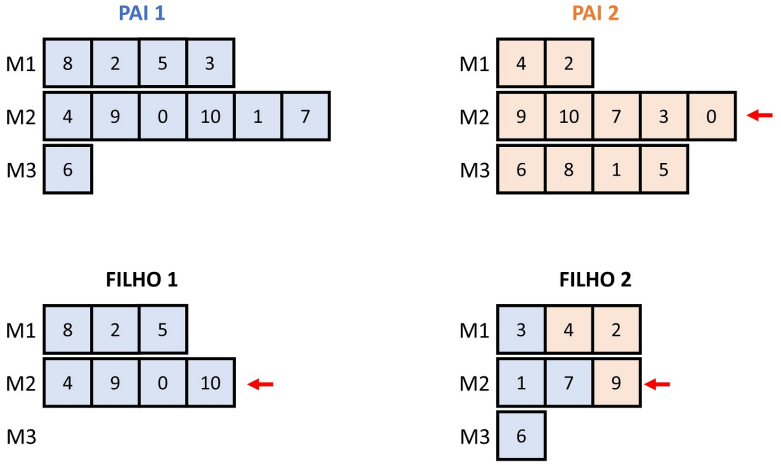
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



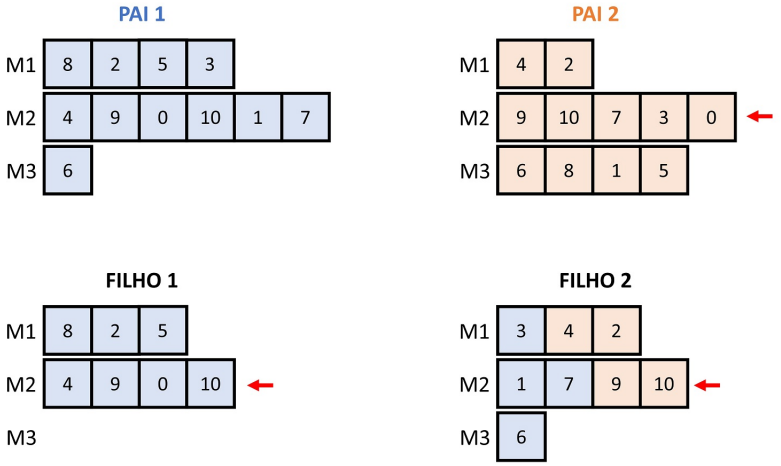
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



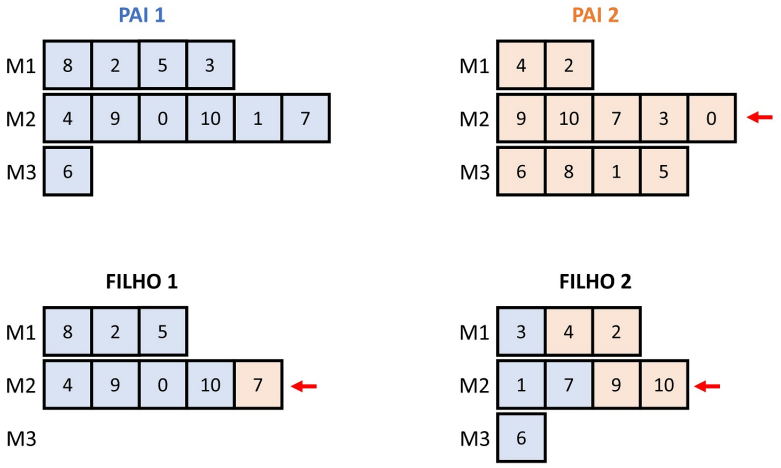
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



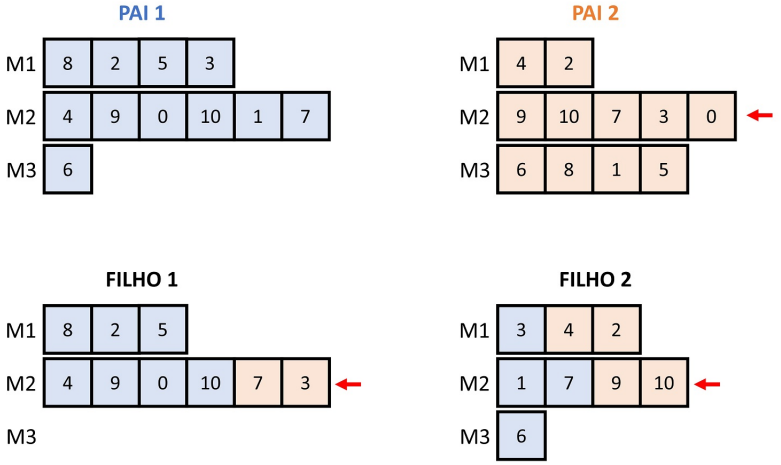
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



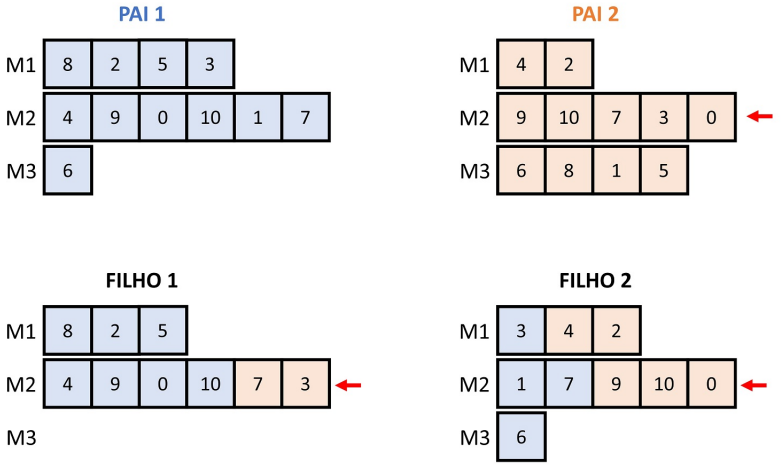
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



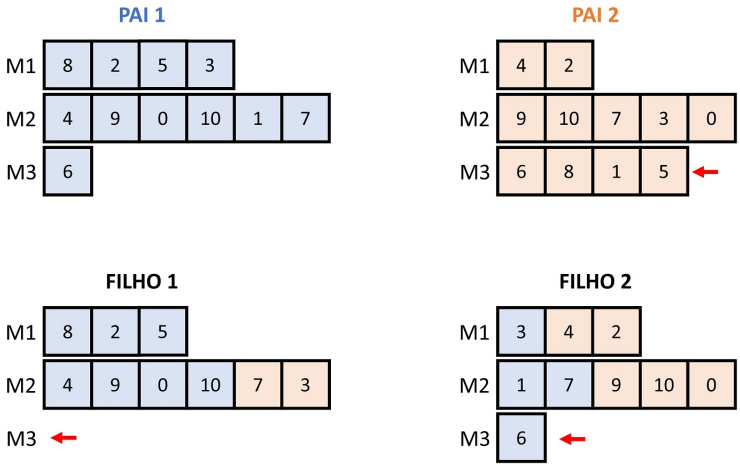
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



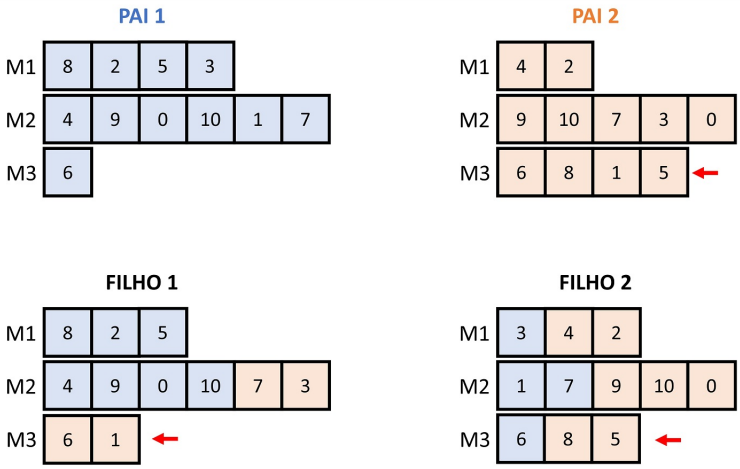
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



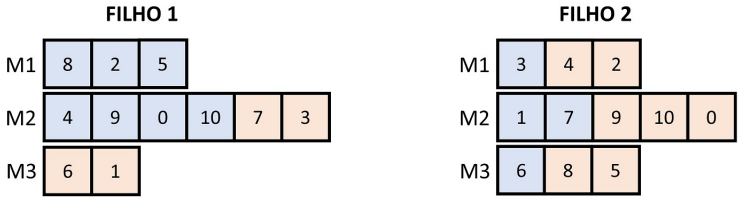
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



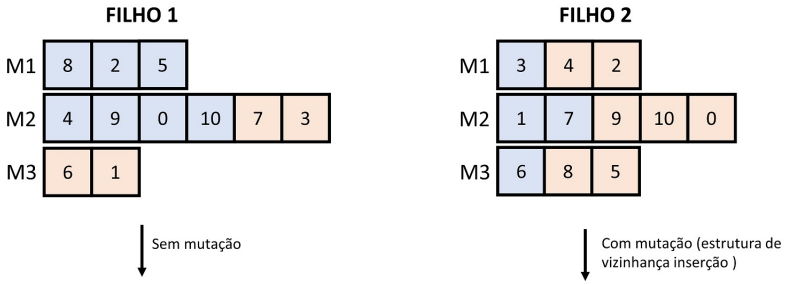
3) Adiciona as tarefas do PAI 2 nos filhos
- Para cada máquina, verifica-se se existe a tarefa do PAI 2 no FILHO 1, se não existir, adiciona a tarefa no FILHO 1, caso contrário, adiciona no FILHO 2



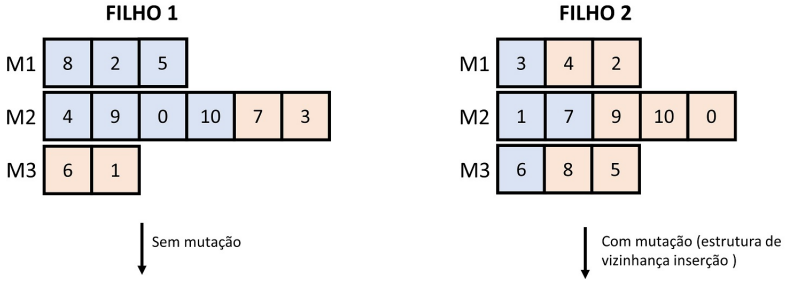
4) Calcula a probabilidade de realizar a mutação em cada filho



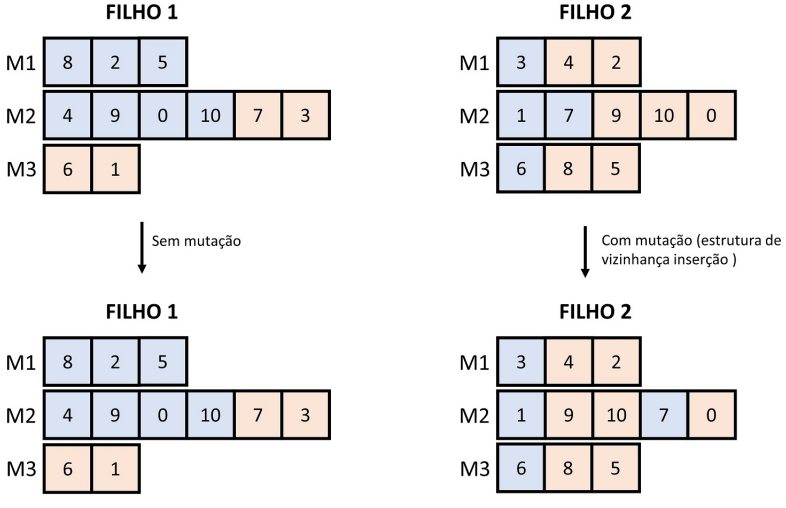
4) Calcula a probabilidade de realizar a mutação em cada filho



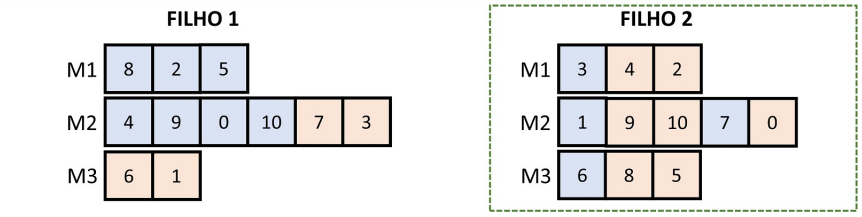
4) Calcula a probabilidade de realizar a mutação em cada filho e qual operação de mutação será realizada (movimento de busca local)



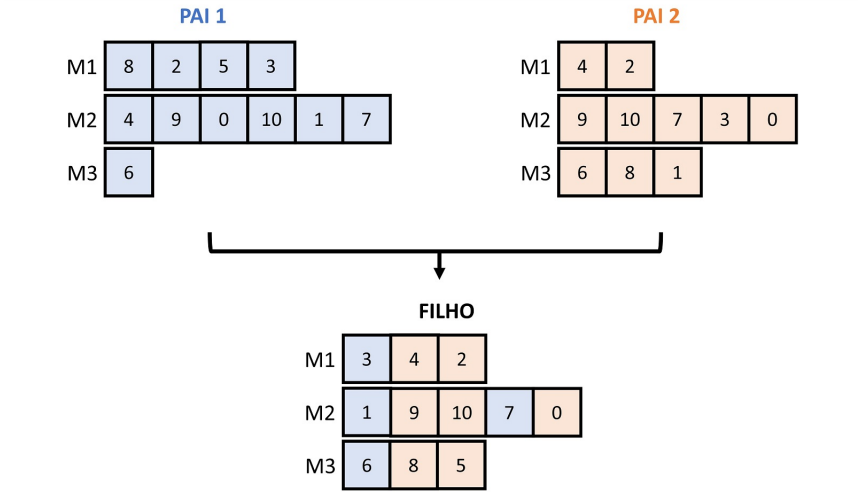
4) Calcula a probabilidade de realizar a mutação em cada filho e qual operação de mutação será realizada (movimento de busca local)



5) Seleciona o melhor filho de acordo com o *Tchebycheff*



6) Retorna o melhor filho



C Publicações

A seguir, são apresentados os artigos publicados referentes à presente tese:

- SANTOS, V. L. A. et al. Multi-objective iterated local search based on decomposition for job scheduling problems with machine deterioration effect. Engineering Applications of Artificial Intelligence, v. 112, p. 104826, 2022. ISSN 0952-1976. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0952197622000926⟩.
- SANTOS, V. L. A. et al. Multi-objective meta-heuristics for a scheduling problem with sequence-dependent machine deterioration. Anais do LIV Simpósio Brasileiro de Pesquisa Operacional; Juiz de Fora-MG. BR. Campinas: Galóa; 2022, v. 54, p. 152781, 2022. Disponível em: ⟨https://proceedings.science/sbpo/sbpo-2022/trabalhos/multi-objective-meta-heuristics-for-a-scheduling-problem-with-sequence-dependent?lang=pt-br⟩.